FÁBIO RODRIGUES DE LA ROCHA

ESCALONAMENTO BASEADO EM INTERVALO DE TEMPO

FLORIANÓPOLIS
2008

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Escalonamento Baseado em Intervalo de Tempo

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

FÁBIO RODRIGUES DE LA ROCHA

Florianópolis, Janeiro de 2008.

Escalonamento Baseado em Intervalo de Tempo

Fábio Rodrigues de la Rocha

Esta Tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica, área de Concentraçãoo em *Automação e sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Prof. Rômulo Silva de Oliveira, Dr.
Orientador

Prof. Kátia Campos de Almeida, Dr.
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Carlos Barros Montez, Dr.
Presidente

Prof. Carlos Eduardo Pereira, Dr.

Prof. Antônio Augusto Medeiros Frohlich, Dr.

Prof. Luiz Cláudio Villar dos Santos, Dr.

Prof. Cristian Koliver, Dr.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

Escalonamento Baseado em Intervalo de Tempo

Fábio Rodrigues de la Rocha

Janeiro/2008

Orientador: Prof. Rômulo Silva de Oliveira, Dr.
Área de Concentração: Automação e Sistemas
Palavras-chave: Tempo Real, QoS, Escalonamento, Modelo de Tarefas
Número de Páginas: xi + 67

Esta tese apresenta um novo modelo de tarefas para expressar requisitos temporais que não podem ser facilmente representados em termos de deadlines e períodos. Neste modelo, tarefas são divididas em segmentos `A`, `B` e `C`. O segmento `A` é responsável por realizar algumas computações e após seu término explicitar o intervalo de tempo dentro do qual o segmento `B` deve executar para cumprir alguns requisitos de aplicação. Finalmente, após a execução de `B` o segmento `C` é liberado para executar. A execução do segmento `B` é válida se realizada dentro daquele intervalo de tempo; caso contrário, sua contribuição pode ser considerada sem valor para sua tarefa. O modelo utiliza funções benefício para indicar quando a ação deve ser executada para obtenção do máximo benefício. Soluções da literatura de tempo real são adaptadas e integradas para produzir uma solução de escalonamento para este problema. Como resultado, foram criadas algumas abordagens (síncronas e assíncronas) desenvolvidas especificamente para o modelo. Testes de escalonabilidade *offline* foram desenvolvidos para cada abordagem. Estes testes, além de um resposta aceita/rejeita, fornecem um limite inferior e superior para a qualidade que será obtida pelo segmento `B` em tempo de execução. No decorrer do trabalho, foram realizadas diversas contribuições à área de tempo real, em específico na área de algoritmos de atribuição de prioridades, redução do pessimismo no tempo de resposta de segmentos não preemptivos e na análise de melhor momento de liberação para os segmentos `B`.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

Time-Interval Scheduling

Fábio Rodrigues de la Rocha

January/2008

Advisor: Prof. Rômulo Silva de Oliveira, Dr.
Area of Concentration: Automation and Systems
Key words: Real-Time, QoS, Scheduling
Number of Pages: xi + 67

This thesis presents a new task model for expressing timing constraints that do not naturally admit expression in terms of deadlines and periods. In our task model, tasks are divided into segments `A`, `B` and `C`. Segment `A` is responsible by performing some computations and eventually adjust the time-interval within the segment `B` should execute to fullfill some application constraints. Finally, after the execution of segment `B`, segment `C` is released to run. The execution of `B` is valid if performed inside that time-interval, otherwise, its contribution may be considered valueless to its task. The model uses benefit functions to specify when an action should be performed for the maximum benefit. We integrate some scheduling approaches from the literature to obtain a possible scheduling solution for our model. As a result, new synchronous and asynchronous scheduling approaches were created specifically to our model. Also, we created new offline feasibility tests targeting each scheduling approach. Besides an accept/reject answer for tasks set, the offline test gives a minimum and maximum expected benefit for segment `B` during run-time. During the course of this work some innovative contributions were made to the real-time literature in areas such as priority assignment algorithms, pessimism reduction in response-time analysis under non-preemptive segments and releasing time analysis to increase the segment `B` benefits.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de abreviaturas e símbolos

RM	*Rate Monotonic*
DM	*Deadline Monotonic*
EDF	*Earliest Deadline First*
T	Período
W	*Worst-Case Execution Time*
D	Deadline
WCRT	*Worst-Case Response Time*
WCET	*Worst-Case Execution Time*
BCRT	*Best-Case Response Time*
rt	*Response-Time*
β	Intervalo entre $[ds, e]$
Ω	Menor intervalo de tempo entre a liberação de dois segmentos B
sf	Fator de deslocamento
$\eta_i(t_1,t_2)$	Número de ativações da tarefa τ_i com liberação e deadline dentro de $[t_1,t_2]$
$f(t_1)$	Tempo de processador utilizado executando interrupções entre $[0,t_1]$
$F(t_1,t_2)$	Tempo de processador utilizado executando interrupções entre $[t_1,t_2]$
A,B,C	Segmentos de Tarefa
Γ	Conjunto de tarefas, $\Gamma = \{\tau_1,\tau_2,\ldots,\tau_n\}$
τ	Tarefa
j	*job* ou ativação
t	Instante de tempo
B_{min}	Limite interior para a liberação de B
$g_{(t_1,t_2)}$	Função de demanda de processador
$G_{(t_1,t_2)}$	Limite superior para a função de demanda de processador
$I_{(B_j,B_i)}$	Interferência que B_j causa em B_i
lp	*Lower priority*
hp	*Higher priority*
mmc	Mínimo Múltiplo Comum
gcd	*Greatest Common Divisor*, maior divisor comum entre dois números
L	Tamanho do *Busy period*
$\overline{x}_{QoS}$	QoS médio para uma determinada atribuição de prioridade
s_{QoS_r}	Desvio padrão do QoS
B_{max}	Limite superior para a liberação de B

H_{prio}	Heurística para selecionar a melhor atribuição de prioridade
U	Utilização
$[s,e]$	Intervalo de tempo onde a função de benefício resulta num valor positivo
ρ	Tamanho do intervalo de tempo $[s,e]$
ψ	Tamanho do intervalo de tempo $[ds,de]$
$[ds,de]$	Intervalo de tempo ideal
$A_{i,j}$	Ativação j do segmento A pertencente a tarefa i
TUF	*Time Utility Function*
JUF	*Joint Utility Function*
GBS	*Generic Benefit Scheduling*
RTA	*Response-Time Analysis*
Φ, ϕ	Maior offset
ϕ_i	*Offset* de uma tarefa τ_i
J	*Jitter*
PIP	*Priority Inheritance Protocol*
PCP	*Priority Ceiling Protocol*
SRP	*Stack Resource Policy*
QoS	*Quality of Service*
$start_{B_j}$	Momento em que B_j começa a executar
end_{B_j}	Momento em que B_j termina de executar
CH	Tempo de computação de uma interrupção
TH	Tempo mínimo entre ativações de uma interrupção
L^*	Ponto em que a equação de $G(0,L)$ cruza a equação de L
$H = mmc(T_1, T_2, \ldots, T_n)$	Hiperperíodo
prio(i)	Prioridade
max	Função máximo entre dois valores

Capítulo 1

Introdução

Um problema de tempo real pode ser definido como aquele no qual a solução depende não somente de uma resposta logicamente correta, mas também correta temporalmente. Isto é, a resposta de um problema estará correta se a resposta estiver logicamente correta e o tempo em que esta é disponibilizada cumpre os requisitos estabelecidos (Liu, 2000).

A solução de um problema de tempo real inicia com um modelo. Modelo é uma representação do problema sobre o qual podem ser aplicados testes e efetuadas simulações até a obtenção de uma solução. Desta forma, é importante que o modelo seja fiel ao problema do mundo físico.

Um exemplo prático é o problema de monitorar um determinado processo físico com um sistema computadorizado. Deve-se amostrar dados de um conversor *A/D* em tempos múltiplos de *T*, realizar algumas operações simples com o valor obtido, testar limites de valores, etc. Em caso de extrapolação dos limites, um alarme deve ser disparado. Além disso, deve-se atualizar o display gráfico do dispositivo para que os valores sejam apresentados na tela corretamente. Esse problema pode ser modelado com uma tarefa periódica que executa a cada período T com instruções para ler o conversor, realizar os cálculos e num caso especial disparar o alarme. Após realizar os cálculos, a tarefa deve armazenar o valor obtido numa posição de memória que e reservada para o vídeo. Uma outra tarefa periódica executa com período de tipicamente $\frac{1}{30}s$ lê os dados da memória do dispositivo e envia para o display. A urgência relativa das tarefas é representada por prioridades e estas podem ser definidas por algoritmos de escalonamento, como por exemplo no caso do algoritmo *Rate Monotonic* (*RM* - onde as prioridades são atribuídas em ordem inversa aos períodos das tarefas) (Buttazzo and Buttanzo, 1997). Podemos nos certificar que estas tarefas terminam antes de um deadline relativo (que para estes propósitos pode ser igual ao período) fazendo o teste de escalonabilidade do *RM*.

Esse exemplo prático ilustra o que é comum na maioria dos problemas de tempo real. As tarefas são liberadas em resposta a um evento (ou à passagem do tempo) e devem terminar sua execução até os seus respectivos deadlines para que o sistema esteja correto temporalmente. O momento em que estas tarefas executam em cada ativação é altamente variável, em face de interferências por outras tarefas de prioridade mais alta ou mesmo bloqueios em virtude de acesso a recursos compartilhados. Independentemente das incertezas quanto ao real momento em que estas efetivamente executam ou

de seu tempo de término, sua execução está correta desde que o tempo entre sua chegada e o momento de término seja menor que seu deadline.

1.1 Problema

Embora muitas aplicações possam ser representadas pelo modelo de tarefas periódicas e deadlines, existem algumas situações onde tarefas possuem requisitos especiais (Ravindran et al., 2005). Em alguns problemas de tempo real, deve-se observar um evento, realizar cálculos e disparar uma tarefa para executar num tempo futuro, o qual é uma função do cálculo realizado anteriormente. Além disso, esse tempo futuro, ou mais especificamente, o intervalo de tempo em que a tarefa deve executar pode estar associado a um benefício. Se a tarefa consegue executar dentro deste intervalo, terá um benefício máximo para o sistema representando o fato de que a operação realizada dentro daquele intervalo de tempo é mais útil. Esse benefício diminui antes e após o intervalo de tempo.

Como exemplo, temos nas Figuras 1.1 e 1.2 um cenário onde uma tarefa T1 é responsável por periodicamente verificar se existem carros numa rodovia. Além disso, T1 deve medir a velocidade dos carros, calcular a sua posição futura e ajustar a execução de uma tarefa T2 para rodar dentro deste intervalo de tempo. Dentro deste intervalo, T2 deve posicionar a câmera nas coordenadas calculadas por T1 e obter imagens para registro. Na ativação representada em 1.1 o intervalo de tempo onde a tarefa deve ser executada (para obtenção do máximo benefício) está representado pelo intervalo $[s,e]$. Nesta figura, T2 é executada antes do momento ideal e a câmera não obtem uma imagem completa do carro. Sua contribuição para a operação de rastreamento de carros é baixa. Já na Figura 1.2 a execução ocorre dentro do intervalo de tempo, resultando numa imagem completa do carro e conseqüentemente no máximo benefício.

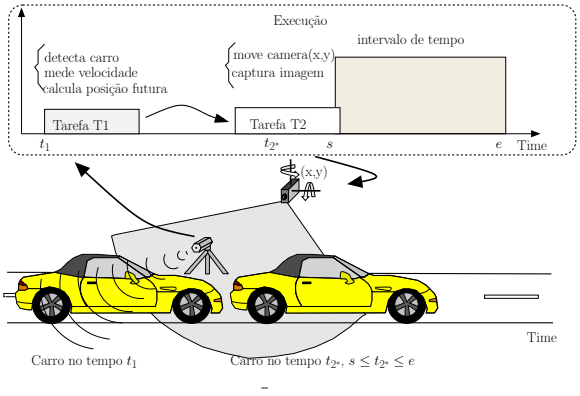


Figura 1.1: Execução com benefício pequeno.

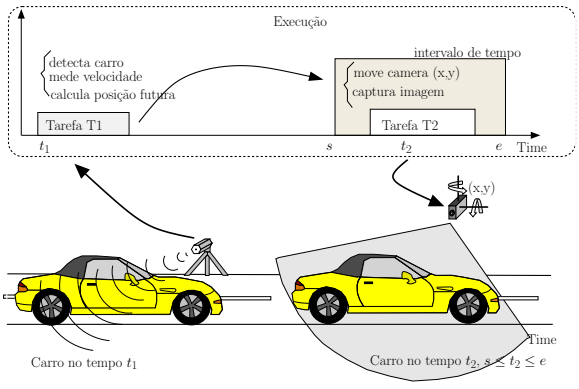


Figura 1.2: Execução com máximo benefício.

Um outro cenário semelhante ocorre em sistemas embarcados onde tarefas podem enviar mensagens utilizando um controlador de protocolo embutido no hardware tal como i^2c, *RS232*, *USB*, *CAN* (Noergaard, 2005). Em muitos microcontroladores de baixo custo, durante a transmissão de dados, a *UCP* (Unidade Central de Processamento) é mantida ocupada movendo dados da memória para a porta do controlador de protocolo e esperando uma resposta ou término da transmissão. Tanto

a tarefa quanto a transmissão precisam ser escalonadas. Além do mais, a transmissão de dados não pode ser interrompida e às vezes deve ser realizada dentro de um intervalo de tempo.

Claramente estes casos de uso não possuem um limite de tempo específico para completar parte de suas computações; no máximo, eles possuem um intervalo de tempo e possivelmente um intervalo de tempo ideal onde a execução de uma parte da tarefa resulta num maior benefício. Assim, o conceito de deadline é inapropriado para modelar estes tipos de aplicações, bem como de um modelo periódico de tarefas. Ainda assim, pela falta de embasamento teórico, estes ainda são implementados com escalonadores convencionais levando à falta de previsibilidade.

Algoritmos bem conhecidos tomam decisões baseando-se na freqüência de chegada das tarefas (por exemplo, Rate Monotonic *RM*), no deadline absoluto (*EDF*) e no deadline relativo (*DM* (Liu, 2000)). Uma melhor solução de escalonamento para o problema seria incluir no escalonador o conhecimento do benefício da tarefa como função do tempo em que ela executa (função benefício).

1.2 Objetivo e escopo do trabalho

O objetivo deste trabalho é fazer uma contribuição à área de sistemas de tempo real, provendo meios de representar e solucionar o problema exemplificado na seção anterior. Inicialmente, define-se um novo modelo para representar o sistema de tarefas, denominado **modelo de tarefas baseado em intervalo de tempo**. Neste modelo, assume-se um intervalo de tempo $[s,e]$ onde o benefício obtido pela execução de uma parte do código da tarefa (**segmento**) é positivo. Dentro deste intervalo de tempo existe um outro intervalo de tempo denominado intervalo de tempo ideal $[ds,de]$, onde a execução do segmento resulta na maior contribuição para a aplicação. O valor obtido pela execução da tarefa é reduzido, antes e após o intervalo de tempo ideal, utilizando funções de benefício (Jensen et al., 1985). Computações realizadas antes e após o intervalo $[s,e]$ podem ser inúteis para os propósitos da aplicação. Ainda assim, em um caso específico podemos ter $ds = s$ e $de = e$ e toda a computação realizada entre s e e obtém o máximo benefício.

Em relação a abordagens de escalonamento, no ambito deste trabalho, considera-se sistemas com apenas um processador. Sendo assim, foram criadas novas abordagens de escalonamento que adaptam outras abordagens clássicas da literatura de tempo real ao problema do intervalo de tempo. Sob o modelo do intervalo de tempo, a escalonabilidade do sistema de tarefas pode ser verificada através de um novo teste de escalonabilidade que, além de fornecer uma resposta aceite/rejeição, fornece o mínimo e o máximo valor de benefício possível para as tarefas. Com base nesta informação, o projetista de sistema pode decidir aceitar o sistema de tarefas ou fazer mudanças. Além disso, apresenta-se uma análise para otimizar o valor de benefício obtido, através de ajustes no momento em que as tarefas devem ser liberadas.

A efetividade do teste de escalonabilidade é avaliada através de comparações com simulações realizadas sobre o mesmo sistema de tarefas, utilizando um simulador especialmente desenvolvido para este propósito. Na seqüência apresenta-se e discute-se alguns trabalhos da literatura de tempo real que se relacionam com o tema desta tese.

1.3 Revisão da Literatura

A abordagem clássica para obter uma execução num instante preciso é através da construção de um sistema *time-driven* (Locke, 1992). Um escalonador *time-driven* possui alta previsibilidade, é de simples implementação e possui grande aceitação em determinados nichos de aplicações. A arquitetura de um sistema *time-driven* é fortemente baseada na regularidade da execução de tarefas periodicas. Consequentemente é mais simples de analisar quanto a escalonabilidade (Kopetz and Grünsteidl, 1994). Infelizmente, este método de escalonar tarefas é conhecido pela sua inerente inflexibilidade quando a escala de execução deve ser alterada (Tokuda et al., 1987).

Em (Burns et al., 2000) os autores apresentam uma visão geral do escalonamento de tarefas baseado em valor e sua capacidade para representar sistemas adaptativos e apresentam um *framework* para escalonamento baseado em valor. Em (Buttazzo et al., 1995) é apresentado um estudo sobre situações de sobrecarga onde as tarefas são compostas por um deadline da tarefa e uma métrica de qualidade. O desempenho do escalonador é avaliado pelos valores cumulativos de todas as tarefas completadas até seus deadlines. O artigo mostra que em situações de sobrecarga, escalonar tarefas com base no seu valor resulta em melhor desempenho.

Em (Liu et al., 1994) é apresentado um modelo para escalonar tarefas compostas por uma parte obrigatória e uma parte opcional que incrementa o benefício obtido da execução da tarefa. Neste modelo, é aceitável que somente as partes obrigatórias sejam executadas. Além disso, a execução da parte opcional não está relacionada com um intervalo de tempo específico, dentro do qual deve executar.

Em (Dey et al., 1996) é apresentado um caso especial de computação imprecisa onde a recompensa obtida aumenta com a execução da tarefa até o seu deadline.

Em (Jensen et al., 1985) é apresentado o modelo de função utilidade (*TUF*) onde existe uma função que associa um benefício à execução da tarefa em relação ao seu tempo de término. O escalonador deve otimizar o benefício acumulado proveniente das ativações das tarefas (*Utility Accrual criteria*). Uma extensão deste trabalho é apresentada em (Wu et al., 2004). Neste trabalho, é apresentado o conceito de *JUF* (*Joint Utility Function*) no qual a utilidade/benefício de uma atividade é especificada em termos do tempo de término de outra atividade. Além disso, eles apresentam a utilidade de uma atividade como função do seu progresso.

Em (Li, 2004) é apresentado um algoritmo de escalonamento heurístico para escalonar tarefas com funções utilidade de formatos arbitrários (*Generic Benefit Scheduling - GBS*). Uma métrica de utilidade potencial dá o benefício esperado para a execução de uma tarefa e todas as tarefas que esta depende.

No problema de controle de chamadas (*call control problem*) apresentado em (Awerbuch et al., 1994) e (Garay et al., 1993) uma seqüência de requisições é feita dinamicamente para alocar um circuito virtual entre dois nós em uma rede. A requisição é composta pelos tempos de início e fim de uso do circuito. Para aplicações como transmissão de vídeo e áudio, a rede tem que garantir uma taxa

mínima de transmissão de bits entre os nós. O objetivo é maximizar o número de chamadas aceitas usando um algoritmo on-line de aceitação.

Em (Lipton and Tomkins, 1994) é apresentado um problema de escalonamento on-line de intervalos no qual um conjunto de intervalos de tempo são apresentados ao algoritmo de escalonamento. Os intervalos de tempo são não-preemptivos, possuem início e fim e devem ser escalonados exatamente num determinado instante de tempo. Como trabalhos futuros, os autores discutem o problema similar no qual os tempos de liberação são mais gerais e onde a tarefa poderia requisitar que um dado intervalo seja escalonado dentro de "x" unidades de tempo. Os intervalos de tempo podem ser deslocados para acomodar outras tarefas. No mesmo contexto de redes de comunicação, em (Goldman et al., 1997) é mostrado um modelo onde os jobs J são não-preemptivos, possuem tempo de chegada a_j, um tamanho $|J|$ e um *delay* máximo w_j. Assim, esses jobs podem ser liberados por um algoritmo de escalonamento on-line durante o intervalo de tempo $[a_j, a_j + w_j)$. Os autores apresentam diferentes modelos de *delays* onde este *delay* pode ser proporcional ao tamanho $|J|$ ou arbitrário.

Em (Chen and Muhlethaler, 1996) os autores mostram uma visão geral sobre escalonamento de tarefas descritas por funções de valor para maximizar o benefício. Ainda que sua apresentação sobre o tema seja genérica, seu modelo de tarefas é limitado a funções cujo valor é sempre incrementado. Além disso, eles utilizam *EDF* não-preemptivo para escalonar tarefas. Durante a execução, a ordem das tarefas do *EDF* é alterada por um algoritmo heurístico para otimizar o benefício. O algoritmo não incorre em perdas de deadlines. A motivação para a escolha de funções cujo valor é sempre positivo é manter o comportamento do *EDF* correto, onde o processador está sempre executando enquanto existirem tarefas prontas a executar. O problema é a impossibilidade de descrever algumas situações do mundo real como as exemplificadas neste capítulo.

No escalonamento *Just in Time Scheduling* uma execução adiantada de uma tarefa é tão ruim quanto uma execução atrasada. O algoritmo de escalonamento tenta minimizar o quanto a tarefa está adiantada (*earliness*) e atrasada (*tardiness*). Muitos artigos sobre escalonamento de *JIT* são encontrados na literatura de pesquisa operacional. Em (Baker and Scudder, 1990) os autores fazem uma revisão da literatura de *JIT* até a data em questão. Em (Hassin and Shani, 2005) é apresentada uma visão geral de problemas *earliness-tardiness* e apresentado um algoritmo polinomial para problemas E/T com penalidades para não execução. Em (Mazzini and Armentano, 2001) os autores apresentam um algoritmo heurístico para escalonar *jobs* minimizando o quanto o job esta atrasado/adiantado para o caso de *jobs* não-preemptivos.

Em (Tindell, 1992) Tindell propôs o uso de *offsets* para controlar a liberação de tarefas. Ele também apresentou a Análise de Tempo de Resposta *RTA* (*Response-Time Analysis*) para um modelo de tarefas com *offsets* objetivando incrementar a escalonabilidade do sistema em sistemas de prioridade fixa. O modelo é estendido para suportar *offsets* dinâmicos em (Gutierrez and Harbour, 1998). Em (Gutierrez and Harbour, 2003), (Pellizzoni and Lipari, 2004) tarefas com *offsets* são consideradas com *EDF*. Em (Pellizzoni and Lipari, 2005) é apresentado um novo algoritmo para verificar a escalonabilidade de transações com offsets e com relações de precedência entre as tarefas que compõem a transação com base no *EDF*. *Offsets* dinâmicos são também apresentados em (Gutierrez and Harbour, 2003). Naquele modelo, subtarefas possuem um *offset* dentro de uma faixa $\phi \in [\phi_{min}, \phi_{max}]$.

O sistema é modelado usando um *offset* mínimo e um *jitter* de liberação J. Cada subtarefa tem um *offset* mínimo ϕ_{min} para controlar a ativação mais cedo da tarefa (liberada depois do final da subtarefa anterior em relação à mesma transação) e um *jitter* igual à diferença entre o pior e o melhor tempo de resposta da subtarefa anterior. Em (Goossens, 2003) é definido o conceito de *offset free systems*, onde os *offsets* não são fixos e o problema é encontrar uma correta associação de offsets tal que o sistema seja escalonável. Neste caso, a precedência entre as tarefas é um requisito difícil de obter.

Em (Crespo et al., 1999) é apresentada a subdivisão de tarefas para melhorar o desempenho em aplicações de controle por computador, reduzindo o atraso computacional. Uma tarefa original τ_i é dividida em três subtarefas, aquisição de dados, computação e atuação. O algoritmo de escalonamento proposto (basedo no *DM*) considera que a precedência é assegurada utilizando diferentes prioridades para cada subtarefa e as faixas de prioridades são agrupadas de acordo com o tipo da subtarefa. Os momentos corretos para liberar a segunda e terceira subtarefa são controlados utilizando-se *offsets*. Os autores também propõem um teste de escalonabilidade para o modelo deles.

Em (Velasco et al., 2003) é apresentado um modelo de tarefas auto-ativado para sistema de controle. Durante a execução, as tarefas podem informar ao escalonador o próximo instante onde devem ser ativadas.

O problema de recursos compartilhados é tratado em (Mok, 1983) assumindo um quanta de processador fixo e não-preemptivo com o tamanho igual ao tamanho da maior seção crítica. Em (Sha et al., 1990) é apresentado o Protocolo de Herança de Prioridade (*PIP-Priority Inheritance Protocol*) e o Protocolo de Prioridade Limite (*Priority Ceiling Protocol PCP*) para prioridades estáticas. Eles foram estendidos respectivamente para o *EDF* em (Spuri, 1995) e (Chen and Lin, 1990). Em (Baker, 1991) é descrito a política de pilha de recurso (*Stack Resource Policy* (*SRP*) para prioridades estática e dinâmica.

1.4 Visão geral e organização da tese

Como será mostrado no texto desta tese, o problema proposto aqui é diferente daqueles mencionados na revisão da literatura e foi necessário propor uma solução específica para ele. Inicialmente apresentamos, no Capítulo 2, uma revisão de alguns conceitos de tempo real que serão utilizados no transcorrer da tese. Na Figura 1.3 apresenta-se uma visão geral da distribuição do restante do conteúdo por capítulos da tese.

Inicialmente, temos um problema do mundo real que não é atendido por soluções existentes na literatura, como nos exemplos apresentados anteriormente (1). Um novo modelo de tarefas foi criado para representar o problema do intervalo de tempo (2) e é descrito formalmente no Capítulo 3, no seu núcleo estão as métricas de qualidade que associam um valor de *QoS* para a execução da parcela de código dentro do intervalo de tempo designado. O modelo permite representar três diferentes modos de execução que resultam em diferentes soluções de escalonamento. Dentre estes três modos de execução, escolheu-se dois modos para uma análise mais profunda. Cada uma destas análises é tratada num Capítulo diferente sobre as abordagens de escalonamento propostas. A primeira delas,

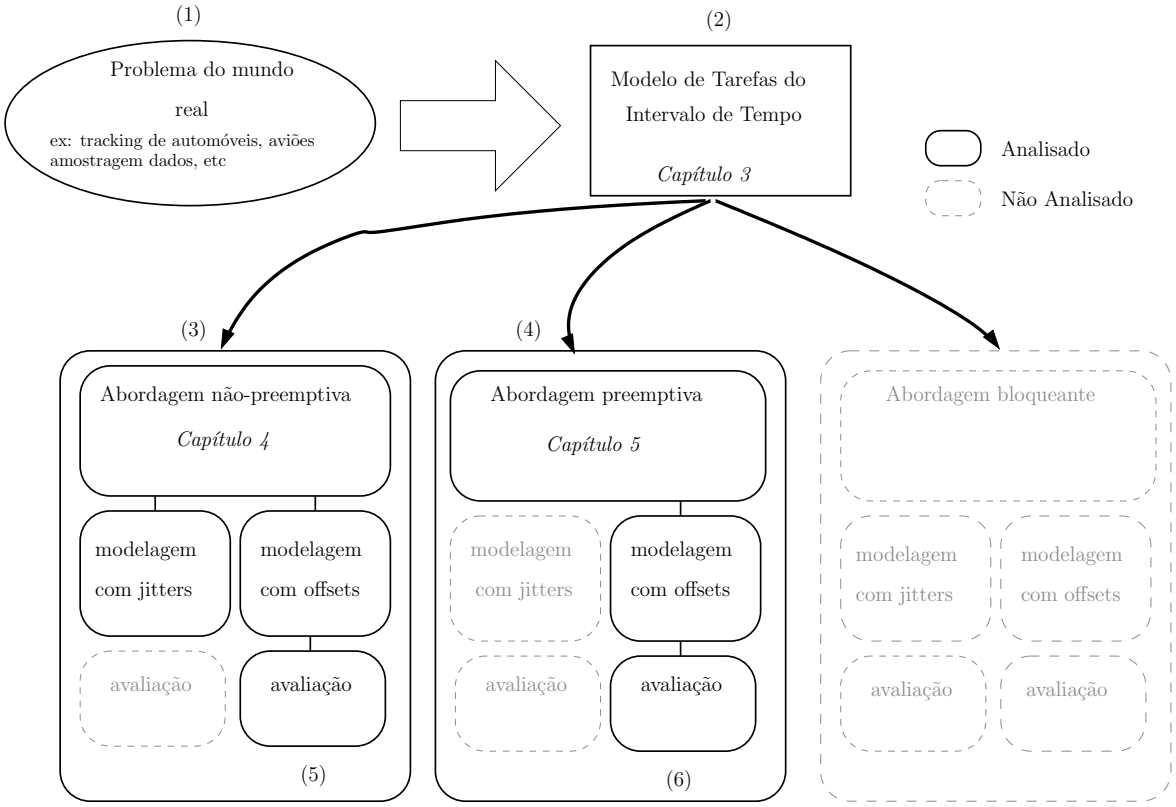


Figura 1.3: Visão geral da Tese

abordagem não-preemptiva (3) (Capítulo 4) na qual existe uma parcela de código que deve executar dentro do intervalo de tempo sem ser preemptada. A segunda abordagem, **abordagem preemptiva** (4) é apresentada no Capítulo 5 e cobre o caso em que as parcelas de código que devem executar dentro do intervalo de tempo podem ser preemptadas por outras de mais alta prioridade. Neste trabalhamos não será apresentada, por limitações de tempo, a abordagem bloqueante, que é uma variação da abordagem preemptiva. As abordagens de escalonamento têm como objetivo apresentar um teste de escalonabilidade que possa verificar se um sistema de tarefas é escalonável ou não (aceite/rejeição). Em nosso caso específico, como trabalhamos com métricas de qualidade, o teste de escalonabilidade fornece, além de uma resposta aceite/rejeição, um limite inferior e um limite superior para a qualidade que pode ser obtida quando da execução de uma tarefa. Com base nesta informação, o projetista pode decidir alterar ou não o sistema.

Como as abordagens de escalonamento resultam em diferentes algoritmos, é de especial interesse verificar o comportamento e os resultados do teste de escalonabilidade. Assim, são realizadas avaliações (5) e (6) sobre um sistema de tarefas, mostrando os valores de qualidade obtidos. Além disso, apresenta-se uma análise para melhorar os valores obtidos para o caso da abordagem não-preemptiva.

O Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos básicos: Sistemas de Tempo Real

Sistemas de tempo real são sistemas nos quais existem requisitos temporais obrigatórios para sua operação. Nestes sistemas o tempo deve ser tratado explicitamente e os requisitos temporais são expressos em geral como um tempo máximo para conclusão de uma tarefa (deadline) (Stankovic, 1992).

Nos sistemas convencionais (não tempo real) só existe a preocupação com o resultado final de um processamento, isto é, se o resultado final está certo ou errado segunda a lógica de uma aplicação. Essa forma de avaliar o resultado final é chamada correção lógica da aplicação. Nos sistemas de tempo real existe interesse, além da correção lógica, também na correção temporal dos resultados; isto é, se valores foram amostrados dentro de intervalos de tempo pré-designados, se a atuação ocorre dentro de intervalos de tempo pr-e-designados e se o processamento foi concluído dentro do tempo pré-designado. Uma aplicação de tempo real estará correta se apresentar correção lógica e temporal.

Sistemas de tempo real são freqüentemente empregados para fazer controle digital, processamento digital de sinais e para lidar com multimídia. Diversas aplicações fazem uso dessas funções, tais como controle de motores, controle de robô, transmissão de vídeo, videogames, equipamentos médicos (tomógrafos, mamógrafos, etc.), controle de freios de carro e equipamentos de manufatura (tais como máquinas de corte e injetoras de plástico). O sistema precisa cumprir os deadlines das aplicações para o seu correto funcionamento.

Nestas aplicações a falha em cumprir o deadline pode ter conseqüências brandas ou graves. Nas aplicações de transmissão de vídeo e videogames, a falha em cumprir os deadlines resultará em redução da qualidade final da aplicação, tal como perda de quadros numa imagem, gráficos sendo mostrados incorretamente, etc. Nesse caso, a falha possui uma conseqüência branda. Nas aplicações de sistemas de freios e equipamentos médicos, as falhas em cumprir deadlines podem causar acidentes ou diagnóstico incorreto de pacientes. A determinação se uma falha possui conseqüências brandas ou graves está diretamente ligada ao valor econômico que a falha resulta e se ela causa risco à vida de pessoas. Na aplicação de controle de um robô, a falha poderia ser branda se o resultado da falha

no funcionamento do robô não envolvesse custos substanciais. Em situações onde o robô é utilizado para exploração espacial, prospecção de petróleo ou montagem de equipamentos existe um alto custo associado, assim, uma falha implica em conseqüências graves.

Em sistemas de tempo real somente a utilização de processadores mais rápidos não garante que os deadlines das aplicações sejam automaticamente cumpridos. Ao utilizar processadores mais rápidos o que ocorre é a redução do tempo de computação das tarefas, enquanto que a previsibilidade temporal continua uma incógnita. Sistemas de tempo real são bastante complexos e não se restringem a *device-drivers* e tratadores de interrupção, sendo uma área de pesquisa intensa (Stankovic, 1988).

Uma das questões mais importantes com respeito a sistemas de tempo real é o escalonamento das tarefas é, em geral, um problema intratável (NP-completo) (Burns and Wellings, 2001). Para que seja compreendido, necessita-se estabelecer alguns conceitos básicos.

2.1 Classificação de sistemas de tempo real

Existem várias formas de classificar sistemas de tempo real. Uma das mais populares tem como base o impacto causado pela perda de deadlines pelas suas tarefas, dividindo os sistemas de tempo real em: (Burns and Wellings, 2001).

Tempo real *soft* - Compreende os sistemas nos quais os deadlines são indicações de quando seria o instante máximo para concluir uma determinada tarefa. Assume que os deadlines podem não ser cumpridos, mas o seu não cumprimento não ocasiona problemas além de uma redução na qualidade dos resultados obtidos.

Tempo real *firm* - Assim como os sistemas com requisitos *soft*, a perda de um deadline não é catastrófica. Difere dos sistemas com requisitos *soft* pois o término de uma tarefa depois de seu deadline não contribui em nada para o sistema.

Tempo real *hard* - Nestes, os deadlines devem ser sempre mantidos, sob pena de sérias conseqüências em termos financeiros, ambientais ou mesmo em vidas humanas.

2.2 Modelos de tarefas

Formalmente, uma tarefa é um conceito abstrato muito utilizado em sistemas de tempo real. Muitas vezes também chamada de processo, uma tarefa é uma unidade de processamento que concorre por recursos no sistema, tal como processador, discos e memória (Silberschatz and Galvin, 1994). Uma aplicação de tempo real é geralmente formada por várias tarefas.

Tarefas podem ser classificadas de diferentes formas, uma das quais é quanto ao momento de ativação. Uma tarefa τ_i é dita periódica (*periodic*) de período T_i se ela é ativada ciclicamente a cada intervalo T_i de tempo e a taxa de chegada de uma tarefa periódica é dada pelo inverso do seu período.

As várias ativações (ou *jobs*) k que uma tarefa pode ter são denominadas instâncias da tarefa τ_i e designadas como $\tau_{i,k}$. Se a tarefa pode ser ativada a qualquer momento, ela é chamada de aperiódica (*aperiodic*) e se a tarefa possuir um tempo mínimo não nulo *min* entre ativações subseqüentes, então ela é chamada de esporádica (*sporadic*) (Burns and Wellings, 2001).

A literatura de tempo real não é unânime sobre esta terminologia. Em (Liu, 2000) a autora faz uma caracterização diferente para tarefas periódicas, aperiódicas e esporádicas. Segundo a autora, uma tarefa periódica possui um período T que é o tempo mínimo entre chegadas sucessivas da tarefa (que aqui consideramos uma tarefa esporádica). Uma tarefa esporádica pode chegar a qualquer momento e tem deadline *hard*. Uma tarefa aperiódica pode chegar a qualquer momento e tem deadline *soft* ou não tem deadline.

Na nomenclatura de tempo real (Buttazzo, 2002), uma tarefa possui um conjunto de características temporais. Algumas dessas são definidas a seguir:

tempo de computação representa quanto tempo uma tarefa necessita até sua conclusão. Como esse tempo em geral não é conhecido, convencionou-se usar o tempo máximo de execução (WCET-*Worst Case Execution Time*) que é uma estimativa de pior caso o tempo de computação de uma tarefa (Puschner and Koza, 1989).

tempo de computação efetivo representa quanto tempo uma tarefa necessita executar em determinada ativação. O tempo de computação efetivo será necessariamente menor ou igual ao WCET.

tempo de início corresponde ao momento em que uma tarefa inicia sua execução, após sofrer interferência de outras tarefas.

tempo de chegada momento em que o escalonador toma conhecimento de uma ativação de tarefa, corresponde ao início do período para as tarefas periódicas.

tempo de liberação é o momento em que a tarefa é colocada na fila de tarefas prontas para executar. Em geral o tempo de chegada coincide com o tempo de liberação, pois a tarefa seria inserida na fila de tarefas prontas logo que ocorre sua ativação. Tais tempos podem não coincidir por características da implementação do escalonador e pela forma de medir o tempo no sistema. Neste caso, existe um atraso de liberação da tarefa denominado *jitter* de liberação (*release jitter*).

tempo de término momento em que a tarefa concluiu. Mesmo que todas as ativações de uma tarefa consigam concluir antes de seu deadline, pode ocorrer que algumas destas terminem antes do que outras. Essa variação em relação ao deadline da tarefa da origem ao *jitter* de saída (*output-jitter*).

criticalidade parâmetro relacionado à gravidade da perda de um deadline, tipicamente se é *soft*, *firm* ou *hard*.

atraso (*lateness*) representa o atraso do término de uma tarefa em relação ao seu deadline. Se a tarefa termina antes do deadline este valor é considerado negativo.

tempo de retardo (*laxity*) tempo máximo que uma tarefa pode ser atrasada na sua ativação e ainda assim completar antes do seu deadline.

tempo de resposta tempo que a tarefa levou desde a sua chegada até o tempo de término. Considerando todas as ativações de uma tarefa, o máximo tempo de resposta é denominado WCRT (*Worst-Case Response Time*).

tempo excedente (*tardiness*) tempo que uma tarefa permanece ativa após o seu deadline.

job ou ativação, instância da tarefa executando. Uma tarefa periódica com período T, chega (e caso não exista *jitter* é liberada) no tempo dado por: $0,T,2T,3T$, etc. ou genericamente $T(k-1)$, para $k \geq 1$.

offset ou fase - tempo contado do momento de chegada até a liberação da tarefa. Para uma tarefa com período T, a k-ésima liberação ocorre no tempo $T(k-1)+offset$, para $k \geq 1$.

precedência em alguns modelos de tarefas pode existir precedência entre tarefas, isto é, restrições quando a ordem de execução de tarefas. Uma tarefa τ_i precede τ_j ($\tau_i \rightarrow \tau_j$) se a tarefa τ_j só pode ser executada depois da tarefa τ_i ter terminado. Geralmente, relações de precedência são representadas como grafos acíclicos onde os nós do grafo são as tarefas e os arcos são as relações de precedência.

exclusão mútua quando tarefas diferentes acessam um recurso compartilhado tal como um dispositivo ou uma estrutura de dados, a execução dessas tarefas pode levar a uma situação na qual o recurso compartilhado ficará num estado inconsistente. Nessa situação, o acesso ao recurso compartilhado deve ser controlado de forma que somente uma tarefa por vez possa utilizá-lo. As demais ficarão bloqueadas esperando a liberação do recurso. Desta forma a consistência é garantida. A parte de uma tarefa que acessa o recurso compartilhado é chamada de seção crítica e deve ser protegida utilizando mecanismos de sincronização (por exemplo, semáforos) para garantir o acesso exclusivo à seção crítica.

inversão de prioridade em sistemas que utilizam prioridades e existe exclusão mútua podem ocorrer situações onde uma tarefa de baixa prioridade está acessando uma seção crítica de código e uma tarefa mais prioritária torna-se disponível. Neste caso, o escalonador decide fazer uma preempção e colocar a tarefa mais prioritária para executar. Caso esta tarefa mais prioritária deseje acessar a seção crítica, ela ficará bloqueada pois uma tarefa de mais baixa prioridade é a detentora do acesso exclusivo a seção crítica. Essa situação caracteriza uma inversão de prioridades, pois a tarefa menos prioritária está bloqueando a mais prioritária.

A Figura 2.1 apresenta três ativações de uma tarefa τ_i com tempo de computação W_i, período T_i e deadline D_i para ilustrar a nomenclatura utilizada.

Com base nesta nomenclatura, podemos descrever uma tarefa periódica τ com a quádrupla $< J,W,T,D >$ onde J é o *jitter* de liberação, W é o tempo de computação, T é o período da tarefa e D é o deadline. Uma tarefa esporádica seria representada pela quádrupla $< J,W,min,D >$ e as tarefas aperiódicas seriam representadas apenas pela tripla $< J,W,D >$.

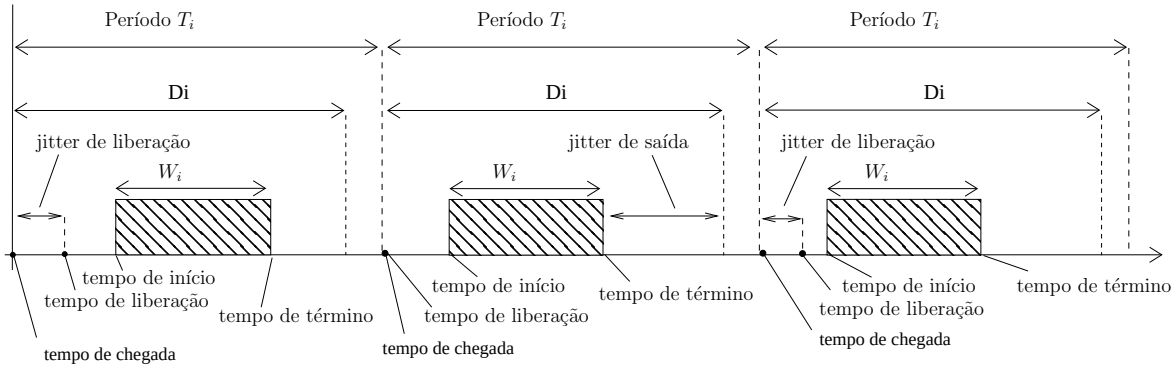


Figura 2.1: Representação de ativações de uma tarefa τ_i.

Num sistema de prioridade fixa, o momento onde todas as tarefas estão prontas para executar é o momento de maior demanda por processador. Ele recebe o nome de instante crítico. Na determinação se um conjunto de tarefas é escalonável (se é possível construir uma escala de execução para todas as tarefas) utiliza-se em geral este momento (Liu, 2000).

2.3 Função utilidade

Na literatura de tempo real, costuma-se representar a utilidade dos dados produzidos por uma tarefa através de uma função de utilidade ou função benefício (*time-value function*) (Jensen, 1993), (Andrews et al., 2002) e (Burns, 1991). Esta função expressa a contribuição dos resultados produzidos em relação ao momento em que estes foram gerados. Em sistemas convencionais (não tempo real) a utilidade da computação não depende do momento em que os dados são produzidos por uma tarefa, pois somente existe interesse se estes estão corretos ou incorretos, tal como na Figura 2.2.

Na Figura 2.3 temos uma função utilidade usualmente associada com requisitos de tempo real *hard*. Nesta, após o deadline, a contribuição dos resultados da tarefa para o sistema tem um valor $-\infty$. Enquanto que em todos os instantes anteriores possui valor máximo, independentemente se este momento ocorreu num tempo infinitesimal após o início do período da tarefa ou num tempo infinitesimal anterior ao seu deadline (Burns and Wellings, 2001).

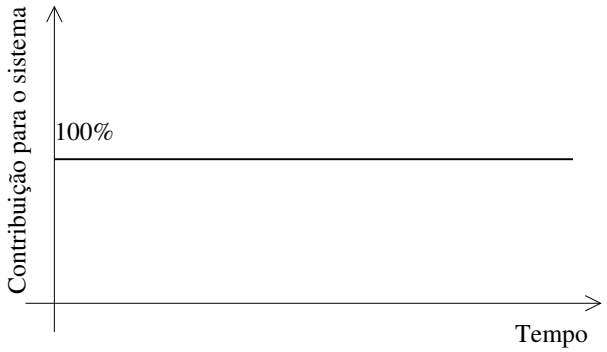


Figura 2.2: Função utilidade num sistema não tempo real.

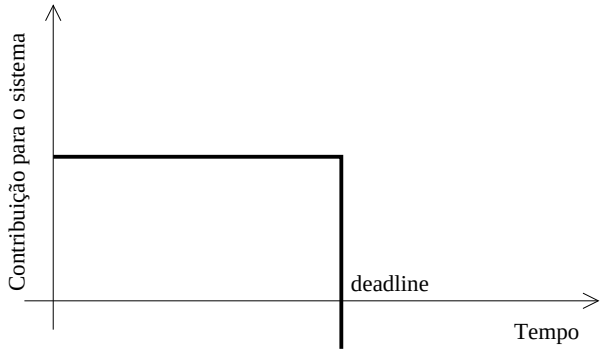


Figura 2.3: Exemplo de função utilidade *hard*.

2.4 Carga estática e carga dinâmica

Num sistema de tempo real assume-se que a carga no sistema é representada pelo somatório de todas as tarefas. Quando todas as tarefas são conhecidas em tempo de projeto e estas são periódicas ou esporádicas, a carga é limitada ou estática. Quando existe alguma tarefa aperiódica, têm-se uma carga dinâmica ou ilimitada (Farines et al., 2000).

2.5 Escalonador

O escalonador (*scheduler*) é responsável por escolher qual tarefa num sistema computacional deve obter acesso a um recurso (tal como processador, disco, memória, etc.). Num sistema computacional existem diversos dispositivos que podem ser gerenciados por um escalonador. Dentre estes recursos, o processador é o que recebe mais destaque.

Os algoritmos de escalonamento são utilizados nos sistemas convencionais (não tempo real) para maximizar o uso do processador, substituindo as tarefas que estão de posse do processador sempre com o objetivo de melhorar o desempenho médio no sistema e/ou promover a justiça na partilha de tempo de processador entre as várias tarefas num sistema.

Nos sistemas de tempo real, o escalonador possui objetivos diferentes. Ele deve selecionar qual tarefa deve obter acesso ao processador para que, idealmente, todas consigam cumprir seus deadlines. Para tanto, ele constrói uma escala com a indicação do momento em que as tarefas devem executar. Um escalonador orientado a prioridades escolhe sempre a tarefa pronta para executar que possua a prioridade mais alta. A regra de atribuição de prioridades tem implicações na utilização máxima do processador e quantidade de trocas de contexto que ocorrerão, estando ligada ao modelo de tarefas utilizado.

2.6 Preemptividade das tarefas

Em alguns modelos de tarefas, a execução de tarefas pode ser fracionada. Uma tarefa a com posse do processador pode ser suspensa pelo escalonador para permitir que uma tarefa mais urgente possa utilizar o processador. A suspensão se dá através de uma troca de contexto. Posteriormente, a tarefa suspensa volta a executar como se não tivesse sido interrompida. Essa suspensão de uma tarefa e substituição por outra é chamada de preempção. Na literatura existem diversos algoritmos de escalonamento que funcionam desta maneira, entre estes o mais popular é o *round-robin* (Silberschatz and Galvin, 1994) que fornece uma fatia de tempo de processador para cada tarefa. Quando a tarefa terminou de executar o seu *quantum* de tempo o escalonador substitui esta tarefa por uma outra obtida de uma lista de tarefas prontas para executar.

Uma tarefa que não pode ser interrompida para dar lugar a outra tarefa é chamada de não preemptável.

2.7 Prioridades

Num sistema onde existem várias tarefas sendo executadas pode-se perceber que existem algumas mais urgentes do que outras. Uma forma para favorecer a execução de um conjunto mais urgente de tarefas em relação a um menos urgente é fazendo que cada tarefa tenha uma prioridade associada. Em nossa convenção, prioridades para tarefas são atribuídas dentro do intervalo $[1,n]$ onde 1 é a mais prioritária e n a menos prioritária. Se uma tarefa possui prioridade 2 ela é mais prioritária do que uma de prioridade 3 e a tarefa 1 é mais prioritária do que a tarefa 2.

Algoritmos de escalonamento utilizam políticas de atribuição de prioridades para tarefas para favorecer determinado conjunto de tarefas em detrimento de outro.

Diferentemente do *round-robin* que usa a passagem do tempo como critério de preempção, algoritmos de escalonamento com prioridades utilizam o valor numérico da prioridade para selecionar a tarefa que deve executar.

Quando uma tarefa está executando e uma tarefa mais prioritária chega, o escalonador preempta a tarefa que esta executando para dar o processador à tarefa de prioridade mais alta. Essa situação caracteriza uma interferência da tarefa mais prioritária sobre a menos prioritária.

Algoritmos de escalonamento podem ser de prioridade fixa (ou estática) quando eles utilizam uma prioridade para as tarefas e esta é mantida fixa durante todo o tempo. Os escalonadores nos quais as prioridades das tarefas mudam durante a execução são chamados de prioridade dinâmica (ou variável). Escalonadores que mantém algumas tarefas com prioridade fixa e outras com prioridade dinâmica são chamados escalonadores mistos.

Como exemplos de algoritmos de prioridade estática temos o *Rate Monotonic (RM)* (Layland and Liu, 1973) e o *Deadline Monotonic (DM)* (Leung and Whitehead, 1982). Como exemplo de prioridade dinâmica tempos o *Earliest Deadline First (EDF)* (Layland and Liu, 1973).

No RM as prioridades são atribuídas às tarefas em tempo de projeto, levando-se em consideração a freqüência de chegada das tarefas. Se uma tarefa ocorre mais freqüentemente, ela possui prioridade mais elevada. No DM as prioridades são atribuídas levando-se em consideração o deadline das tarefas. Quanto menor o deadline relativo, maior a prioridade da tarefa.

No EDF a atribuição de prioridades é feita em tempo de execução baseada no deadline absoluto das tarefas. Uma tarefa recebe uma prioridade maior quanto mais próximo está o seu deadline absoluto. Como num sistema de tarefas periódicas o deadline absoluto depende da instância atual da tarefa, o EDF é um método de atribuição dinâmica de prioridade.

2.8 Escalonamento estático e escalonamento dinâmico

No escalonamento estático, as decisões de escalonamento são tomadas com base em parâmetros fixos associados às tarefas antes de sua execução. Pode-se considerar nesta categoria escalonadores como o *clock-driven* ou *time-driven* (Cheng et al., 1988) no qual o escalonador realiza em tempo de projeto (*offline*) a construção da escala de execução das tarefas. Este tipo de escalonador especifica o momento em que cada tarefa deve executar e especifica que a quantidade de tempo de processamento alocado para uma tarefa deve ser igual ao tempo máximo de computação desta (WCET). Neste, as tarefas terminarão no seu deadline e todos os deadlines serão cumpridos. Também incluí-se nesta categoria de escalonamento os escalonadores com prioridade fixa.

No escalonamento dinâmico as decisões de escalonamento são tomadas com base em parâmetros que podem mudar durante a evolução do sistema, tais como prioridades e eventos (chegada de tarefas novas). Nos sistemas chamados *event-driven* ou *event-triggered* todas as atividades do sistema são tomadas em resposta a eventos externos que podem ocorrer a qualquer instante.

2.9 Classificação das abordagens de escalonamento

As abordagens de escalonamento podem ser classificadas de diferentes formas. Na literatura de tempo real não existe uma classificação padrão. Utilizando os critérios vistos anteriormente, pode-se estabelecer uma classificação arbitrária das abordagens de escalonamento, conforme ilustrado na Figura 2.4 (Farines et al., 2000).

Um dos critérios refere-se à previsibilidade que é oferecida: previsibilidade em tempo de projeto e abordagens de melhor esforço que não fornece garantia de cumprimento de deadlines (no máximo fornecem uma previsibilidade probabilista).

2.9.1 Garantia em tempo de projeto

Na garantia em tempo de projeto existe a premissa de uma carga limitada, conhecida em tempo de projeto, de reserva de recursos para a execução de todas as tarefas no pior caso (caso de pico).

É possível calcular precisamente o que o processador deve fazer a cada instante. Uma estratégia é criar uma grade (ou escala de execução) que descreve a execução de cada tarefa. Um programa de controle fica responsável por repetir ciclicamente essa grade (executivo cíclico). Qualquer conflito por recursos, precedência de tarefas, etc. são resolvidos em tempo de projeto na criação da grade de execução. Com a simples inspeção da grade, é possível verificar se todas as tarefas conseguirão cumprir os seus deadlines.

Outra técnica consiste em atribuir a cada tarefa uma prioridade. Em tempo de projeto é realizado um teste de escalonabilidade para verificar se as tarefas são escalonáveis (se todas as tarefas conseguirão manter o seu deadline). Em tempo de execução, um escalonador preemptivo faz com que a tarefa com prioridade mais alta seja selecionada para execução.

2.9.2 Melhor esforço

Na abordagem de melhor esforço existe a possibilidade de haver sobrecarga, situação na qual não é possível executar todas as tarefas cumprindo-se seus deadlines. Na situação de sobrecarga, o escalonador deve utilizar uma política para flexibilizar a execução de algumas tarefas. Existem, basicamente, quatro formas (Farines et al., 2000):

- eliminar completamente algumas tarefas;
- flexibilizar o tempo de execução de algumas tarefas;
- flexibilizar o prazo de execução de algumas tarefas;
- flexibilizar o período de algumas tarefas.

2.9.3 Garantia dinâmica

Na garantia dinâmica (Ramamritham and Stankovic, 1994) o sistema tenta manter o deadline das tarefas mas não oferece garantias que os deadlines sempre serão alcançados. Utiliza-se um teste de aceitação para verificar se uma tarefa que entra no sistema é escalonável em conjunto com as tarefas já existentes na fila de tarefas prontas. Se o conjunto formado pela nova tarefa e pelas tarefas já existentes no sistema não puder ser escalonado, a tarefa recém chegada será descartada, mantendo as demais. A garantia dinâmica pode ser aplicada tarefa a tarefa ou ativação a ativação, sendo o segundo caso o mais comum.

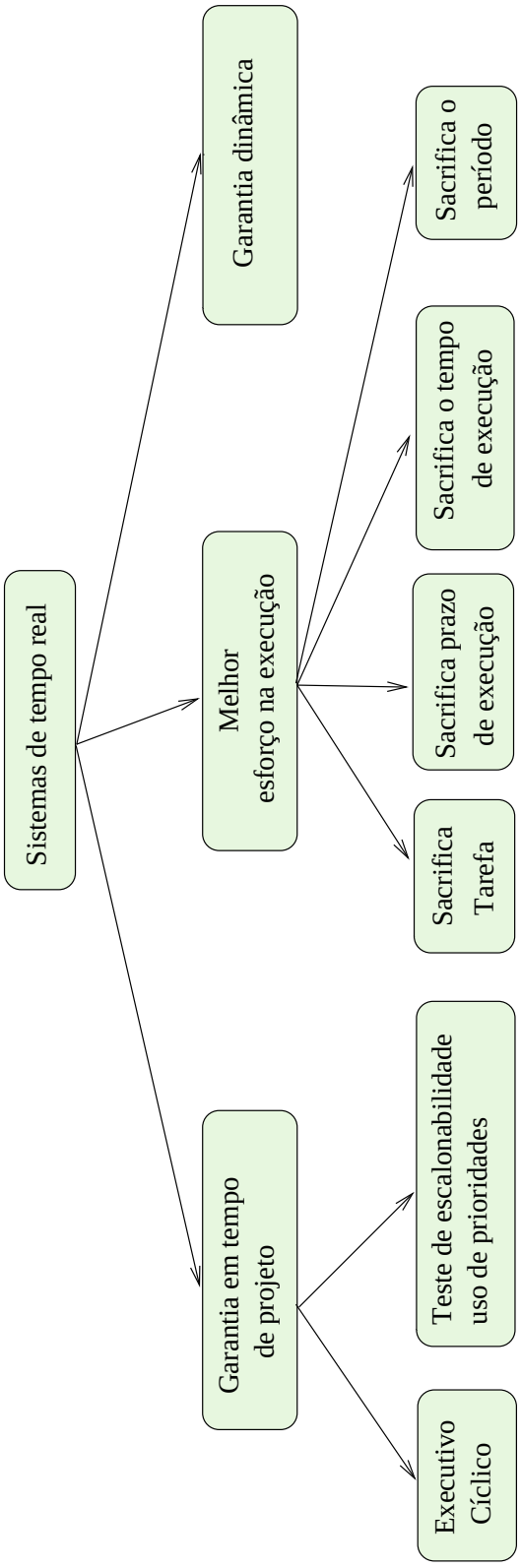


Figura 2.4: Classificação das abordagens de escalonamento.

Capítulo 3

Modelo de Tarefas Baseado em Intervalo de Tempo

As aplicações apresentadas no Capítulo 1 fazem parte de uma classe de problemas do mundo real que não podem ser corretamente representadas com os atuais modelos de tarefas de tempo real. Neste capítulo, apresenta-se o **modelo de tarefas baseado em intervalo de tempo** como forma de representar corretamente aquelas aplicações e permitir que, posteriormente, desenvolva-se um algoritmo de escalonamento adaptado ao novo modelo de tarefas.

3.1 Definições

O modelo de tarefas baseado em intervalo de tempo é conjunto de tarefas τ composto por τ_i, $i \in \{1 \ldots n\}$. Tarefas τ_i são caracterizadas pelos seguintes atributos: tempo de execução de pior caso W_i, período T_i e um deadline D_i. Considera-se que $T_i = D_i$. Cada τ_i consiste em uma infinita seqüência de *jobs* $(\tau_{i1},\ldots,\tau_{ij},\ldots)$, o j^{th} *job* da tarefa τ_{ij} chega (*arrival time*) no tempo $(j-1)\cdot T_i, j \geq 1$ e deve terminar até o tempo $(j-1)\cdot T_i + D_i$ ou até que uma falha temporal ocorra. Define-se como **segmento** um grupo seqüencial de instruções dentro de τ_i (Figura 3.1).

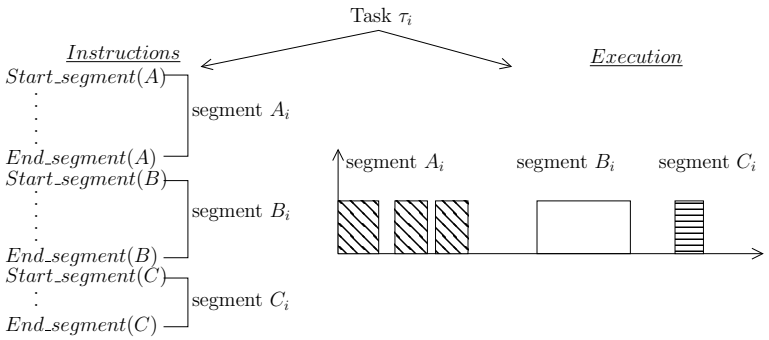


Figura 3.1: Tarefa τ_i com divisão clara entre seus segmentos.

Uma tarefa τ_i é composta por três segmentos chamados A_i, B_i e C_i. Denota-se o primeiro índice do segmento como representativo da tarefa em questão e segundo índice como o *job* (ou ativação) referenciado. Desta forma, o primeiro *job* do segmento A_i é chamado A_{i1}, o segundo *job* é A_{i2} e assim por diante para todos os segmentos. O pior tempo de execução de A_i é W_{A_i}, o de B_i é W_{B_i} e o de C_i é W_{C_i}. A soma do pior tempo de execução de todos os segmentos é igual ao pior tempo de execução da tarefa τ_i (W_{A_i}+W_{B_i}+W_{C_i}=W_i). É assumido que existe uma relação de precedência entre os segmentos $A_i \prec B_i \prec C_i$, conforme mostra a Figura 3.2.

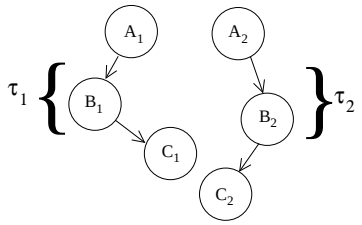


Figura 3.2: Precedência entre os segmentos.

A execução dos segmentos A_i, B_i e C_i é sujeita ao deadline D_i da tarefa τ_i. O segmento A_i é responsável por realizar suas computações e pode requerer ou não a execução do segmento B_i, responsável por **realizar operações em dispositivos**. Desta forma, o tempo de chegada do segmento B_i é determinado em tempo de execução pelo segmento A_i. Caso a execução do segmento B_i seja requerida, o segmento C_i (que é um código de finalização da tarefa) deve ser executado. Assim, mesmo que a execução do segmento A_i seja periódica com período T_i, os segmentos B_i e C_i são esporádicos.

Caso B_i e C_i não sejam requisitados para executar, o segmento A_i poderá executar até o deadline D_i. Caso contrário, logo após o segmento B_i concluir sua execução, o segmento C_i será liberado para executar.

Considera-se o escalonamento num sistema de apenas um processador, sendo assim, os segmentos não podem sobrepor-se no tempo (não é possível a execução paralela de segmentos).

3.2 Métrica de Qualidade de Serviço

A execução do segmento B_{ij} está também sujeita a um **intervalo de tempo** $[s_{i,j}, e_{i,j}]$ o qual é definido pelo segmento A_{ij} em tempo de execução e pode mudar para cada *job* j, isto é: o segmento B_{ij} deve executar dentro deste intervalo de tempo para gerar um benefício positivo. O tamanho de $[s_{i,j}, e_{i,j}]$ é constante e chamado ρ_i. Dentro do intervalo de tempo $[s_{i,j}, e_{i,j}]$, existe um **intervalo de tempo ideal** $[ds_{i,j}, de_{i,j}]$ com tamanho constante, denominado ψ_i, no qual a execução do segmento B_{ij} resulta no maior benefício para τ_i ($W_{B_i} \leq \psi_i \leq \rho_i$).

A Figura 3.3 mostra dois *jobs* da execução da tarefa τ_i (seção superior) e a função benefício do segmento B_i (seção inferior). As Figuras 3.4 e 3.5 foram criadas para representar requisitos comuns de aplicações e, desta forma, também representam diferentes requisitos de tempo real. A Figura 3.4

representa um requisito de sistemas de tempo real **cumulativo** onde o benefício é reduzido do máximo (dentro do intervalo de tempo ideal) para zero nos limites do intervalo de tempo. O benefício obtido pela execução do segmento B_{ij} é o somatório dos benefícios obtidos em cada fração de código que B_i executa dentro da ativação j. Somente as frações que executam dentro do intervalo de tempo $[s,e]$ possuem valor a contribuir para a execução de $B_{i,j}$. A Figura 3.5 representa um requisito de tempo real **rígido** onde o segmento B_i deve executar dentro do intervalo de tempo $[s_j,e_j]$, caso contrário, o benefício será $-\infty$, representando uma conseqüência catastrófica. A escolha de uma função em particular para uma tarefa é um requisito da aplicação a qual também determinará os valores de $s_{i,j}$,$e_{i,j}$,$ds_{i,j}$ e $de_{i,j}$.

Nestas figuras, o eixo y representa o benefício obtido v e o eixo x o tempo de ativação t. O segmento B_{ij} é apresentado como executando com o seu pior tempo de execução (W_{B_i}), iniciando em $start_{bj}$ e terminando em end_{bj}. A função benefício $v(t)$ em função do tempo é dada pelas equações em cada figura. Na Equação 3.1 o *QoS* é mostrado como o benefício obtido na execução do segmento B_{ij} dentro do intervalo de tempo. A equação resulta em um valor entre $[0,100\%]$ e representa o percentual do máximo benefício. O máximo benefício é somente alcançado quando B_i executa todo o seu código dentro do **intervalo de tempo ideal** $[ds_{i,j},de_{i,j}]$. O objetivo é maximizar o *QoS* para cada execução de B_i. A execução do segmento B_i não é sempre necessária e para os casos em que B_i não é requisitado não existe valor de benefício para contabilizar.

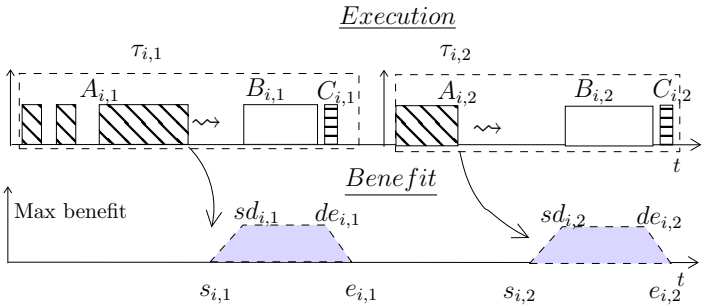
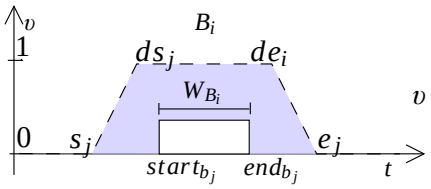
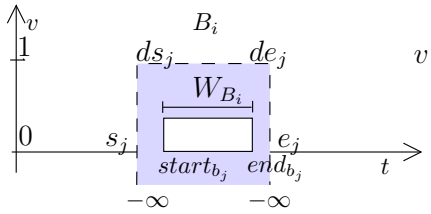


Figura 3.3: Modelo de tarefa mostrando dois *jobs* da tarefa τ_i e uma função de *QoS* Firm.



$$v(t) = \begin{cases} 0, & t < s_j \text{ or } t > e_j \\ 1, & ds_j \le t \le de_j \\ 1 - \left(\frac{ds_j - t}{ds_j - s_j}\right), & s_j \le t < ds_j \\ 1 - \left(\frac{t - de_j}{e_j - de_j}\right), & de_j < t \le e_j \end{cases}$$

Figura 3.4: Benefício para um sistema de tempo real cumulativo.



$$v(t) = \begin{cases} -\infty, & t < s_j \text{ or } t > e_j \\ 1, & s_j \le t \le e_j \end{cases}$$

$$s_j = ds_j \qquad e_j = de_j$$

Figura 3.5: Benefício para um sistema de tempo real rígido.

$$QoS(B_{i,j}, start_{B_{i,j}}, end_{B_{i,j}}) = \frac{\int_{start_{B_{i,j}}}^{end_{B_{i,j}}} v(t)\,dt}{end_{B_{i,j}} - start_{B_{i,j}}} \cdot 100 \tag{3.1}$$

3.2.1 Outras Métricas de Qualidade Possíveis

Além das métricas de qualidade apresentadas na seção anterior, pode-se utilizar variações daquelas métricas para representar diferentes requisitos de aplicação (Figuras 3.6,3.7,3.8 e 3.9).

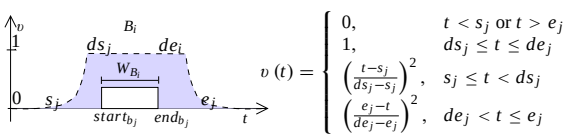


Figura 3.6: Equação para benefício *suave*.

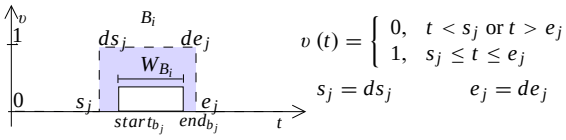


Figura 3.7: Equação para benefício *brusco*.

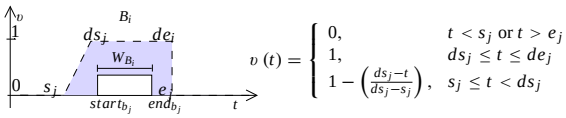


Figura 3.8: Equação para benefício *linear anterior*.

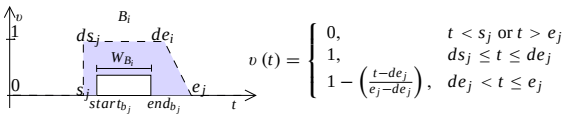


Figura 3.9: Equação para benefício *linear posterior*.

3.3 Modos de execução

Além dos requisitos de tempo, o problema do intervalo de tempo pode também apresentar requisitos de acesso exclusivo durante a execução do segmento B dentro do intervalo ideal. A natureza dos recursos sob o qual o segmento B realiza operações pode impor requisitos de controle de acesso para resguardar que a operação manterá o recurso num estado consistente. É assumido que durante a execução dentro do intervalo de tempo, o processador é mantido ocupado. Os modos de execução do segmento B serão descritos a seguir.

3.3.1 Modo de execução preemptivo

A execução do segmento B_i de uma tarefa τ_i pode ser interrompida por outro segmento B_j da tarefa τ_j com maior urgência (prioridade). Por exemplo, tarefas τ_i e τ_j podem compartilhar o mesmo sensor para uma operação de aquisição de dados. O *QoS* de τ_i é o *QoS* cumulativo obtido por B_i enquanto executando dentro do intervalo de tempo. A Figura 3.10 mostra uma execução de segmentos num sistema escalonado pelo *EDF*. Como os segmentos B podem ser interrompidos por segmentos de prioridade mais alta, em t_1 o segmento A_j chega e ganha o processador. Em t_2 o segmento B_k chega mas sua prioridade é mais baixa do que o segmento que está executando. Em t_3 quando A_j termina o

segmento de maior prioridade é B_k e este ganha o processador, executando até o seu fim em t_4 quando B_i volta a executar.

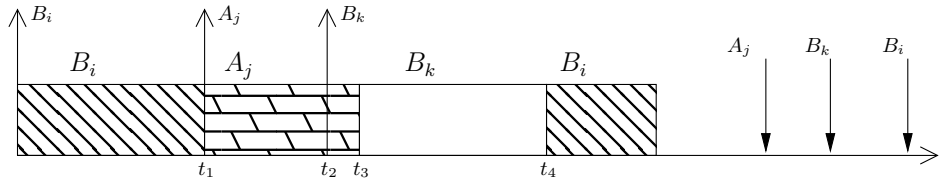


Figura 3.10: Modo de execução preemptivo.

3.3.2 Modo de execução bloqueante

A execução de B_i da tarefa τ_i pode ser interrompida por outra tarefa τ_j mas, neste caso, B_j não pode executar enquanto B_i não tiver terminado (a execução dos segmentos B é serializada). Esta restrição é importante para manter a consistência em casos onde tarefas τ_i e τ_j devem acessar um sensor direcional ou outro dispositivo no qual é necessário manter o acesso mutuamente exclusivo e os custos de uma operação de *rollback*[1] são muito altos. Desta forma, a execução do segmento B_i é similar a execução de uma tarefa num sistema com recursos compartilhados. Na Figura 3.11 é mostrada a execução dos segmentos num sistema escalonado pelo *EDF*. Em t_1 o segmento B_i é interrompido pela chegada do segmento A_j. No tempo t_2 o segmento B_k chega para ser escalonado mas possui prioridade mais baixa do que o segmento atualmente executando. Em t_3 o segmento A_j termina, porém, o segmento B_k mesmo possuindo prioridade mais alta não pode executar em virtude de acessar o mesmo recurso que B_i acessa. Neste caso, B_i é posto para executar até o seu final em t_4 quando, finalmente, B_k ganha o processador.

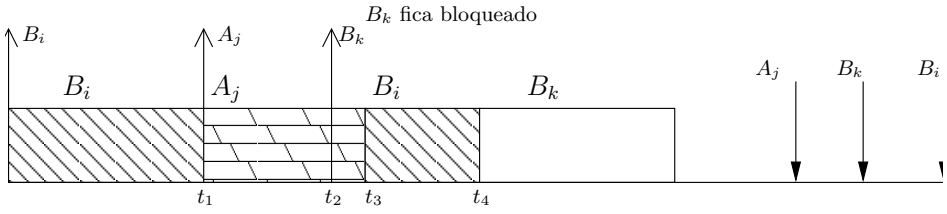


Figura 3.11: Acesso exige bloqueio.

3.3.3 Modo de execução não-preemptivo

A execução do segmento B_i de uma tarefa τ_i não pode ser interrompida por outra tarefa τ_j por apresentar requisitos de execução estritos e acesso mutuamente exclusivo do dispositivo. Neste caso, o início da execução de B_i pode ser postergado, mas uma vez iniciado não pode ser interrompido. Por exemplo, em problemas de rastreamento de objetos em tempo real e controle de equipamentos industriais, sensores/atuadores não podem ser compartilhados entre tarefas enquanto existe uma operação em andamento. Além disso, para assegurar o correto comportamento temporal, a operação não pode

[1] Operação na qual o sistema é colocados no exato estado anterior ao início de uma operação cancelada

ser interrompida. Na Figura 3.12 é mostrada a execução dos segmentos num sistema escalonado pelo *EDF*. Em t_1 o segmento A_j está pronto para executar, mas como neste modo, os segmentos B executam sem ser perturbados, B_i continua executando. Em t_2 o segmento B_i termina e A_j ganha o processador. Em t_3 o segmento B_k chega mas possui prioridade menor que o segmento executando, sendo postergado. Em t_4 o segmento A_j termina e B_k ganha o processador.

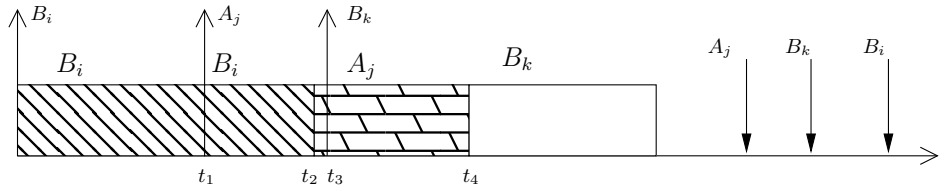


Figura 3.12: Preempção não permitida.

3.3.4 Modos de execução: discussão

Na descrição dos modos de execução, os segmentos B foram apresentados como sendo escalonados pelo *EDF* (como os segmentos A e C). Dessa forma, os segmentos B podem ser preemptados por segmentos A ou C de prioridade relativa mais alta. Uma outra possibilidade é fazer uma diferenciação hierárquica de importância entre os segmentos. Nas abordagens de escalonamento, apresenta-se uma diferenciação entre importância dos segmentos através da forma de atribuir prioridade e do método de escalonamento.

3.4 Abordagens de escalonamento propostas

Os diferentes modos de execução para o segmento B_i vistos anteriormente impõem diferentes requisitos de controle de acesso para um recurso compartilhado. Esses requisitos resultam em algoritmos de escalonamento diferenciados sob o ponto de vista da análise de escalonabilidade, sendo alguns modos de execução penalizados. Claramente, o modo de execução não preemptivo é mais restritivo do que os demais e penaliza a escalonabilidade do sistemas de tarefas. Nos próximos capítulos algumas abordagens clássicas da área de escalonamento de tempo real são integradas e convenientemente modificadas para representar o modelo de intervalo de tempo.

Na Figura 3.13 temos uma visão hierárquica do que será apresentado nos próximos capítulos. No retângulo da esquerda temos os modos de execução apresentados nesse capítulo e no retângulo da direita estão as abordagens de escalonamento com destaque para as implementações utilizadas. No capítulo 4 será apresentada uma abordagem para lidar com subtarefas B que não admitem preempção e são implementadas tanto com relações de *jitter* quanto relações de *offsets*. No Capítulo 5 será apresentada uma abordagem para lidar com uma situação em que subtarefas B_i podem sofrer preempção por subtarefas B_j e onde as subtarefas são modeladas com *offsets*.

Para propósitos de implementação, é natural representar os segmentos de tarefas como subtarefas. Assim, mapeia-se todos os segmentos de tarefas τ_i em subtarefas, mantendo os mesmos nomes A_i, B_i,

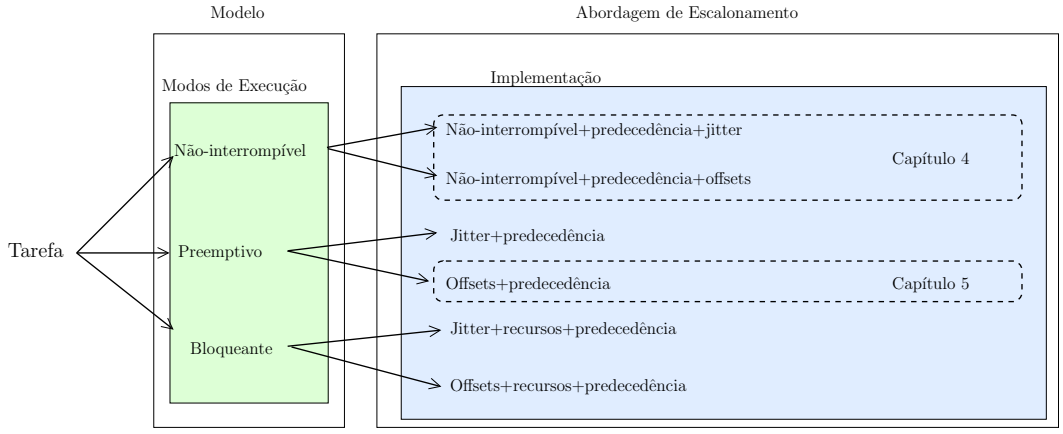


Figura 3.13: Visão geral deste capítulo.

C_i. Subtarefas A_i e C_i são escalonadas utilizando-se *EDF* preemptivo por sua capacidade de explorar toda a largura de banda do processador (Buttazzo, 2005). Para facilitar a análise da escalonabilidade, assume-se que as subtarefas B possuem prioridades fixas, mais altas do que as prioridades de A e C e, dessa forma, somente podem ser preemptáveis por outras subtarefas B com prioridades mais altas (Figura 3.14).

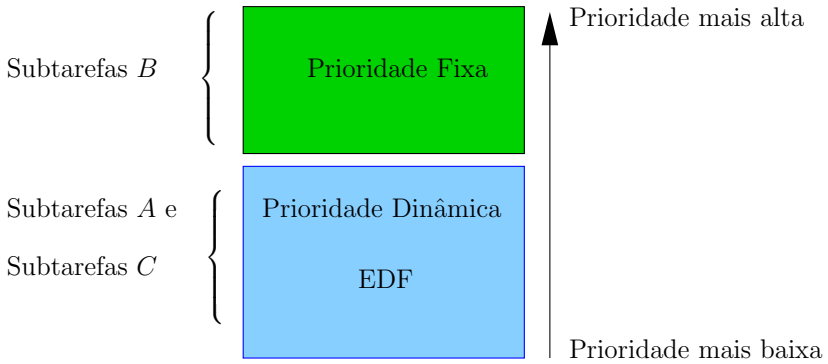


Figura 3.14: Distribuição de prioridades.

3.5 Desafios

O novo modelo de tarefas possui alguns desafios que devem ser solucionados para criar uma abordagem de escalonamento conveniente.

particionamento de deadline Em muitos problemas de escalonamento, o interesse principal é assegurar que os deadlines das tarefas serão respeitados. Nestes casos, os deadlines e os períodos são requisitos da definição do problema com uma correspondência direta no mundo físico. Quando uma tarefa τ_i pode ser dividida em subtarefas, por exemplo: em um sistema multiprocessado, cada subtarefa possui seu próprio deadline e a última subtarefa tem que respeitar o deadline previamente definido para τ_i, neste caso um deadline fim-a-fim D_i. Os deadlines

internos são associados a subtarefas através de uma regra de particionamento de deadline. Na definição do problema do intervalo de tempo o segmento B_i deve iniciar dentro de uma janela de tempo ajustada por A_i. O mínimo tempo para liberar B_i é um requisito do problema e este valor pode ser utilizado como um deadline para o segmento anterior. Assume-se um limite inferior e um limite superior para a liberação do segmento B_i $[Bmin_i, Bmax_i]$ e ajusta-se o deadline $D_{A_i} = Bmin_i$ e o deadline de $D_{B_i} = Bmax_i + \rho_{B_i}$. O deadline de C_i é o próprio deadline da tarefa τ_i e seu tempo de chegada é o deadline B_i (D_{B_i}) (Figura 3.15).

precedência Durante a execução de tarefas no *EDF*, a correta precedência pode ser assegurada através dos deadlines de cada subtarefa (Blazewicz, 1976). Considerando o deadline original da tarefa τ_i como D_i, é possível associar um novo deadline para cada subtarefa tal que $D_{A_i} < D_{B_i} < (D_{C_i} = D_i)$. Desta forma, a ordem de execução estará conforme o requisito de precedência.

liberação das subtarefas É necessário refletir no teste de escalonabilidade que uma subtarefa deve ser liberada somente depois de um tempo predeterminado. Este controle pode ser obtido através da modelagem de *jitters* para postergar a liberação de uma subtarefa ou através de *offsets*. Um teste offline de escalonabilidade, orientado a *jitters*/*offsets*, pode cumprir estes requisitos e garantir que o sistema de tarefas será capaz de executar sem perder nenhum deadline. Um suporte em tempo de execução garantirá que os tempos de liberação serão obedecidos.

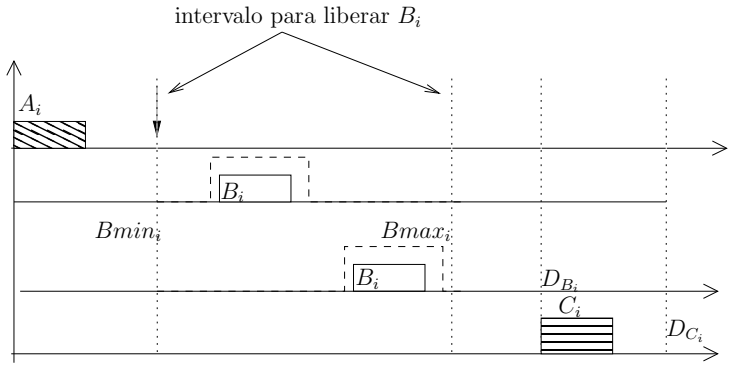


Figura 3.15: Limites para liberação de B_i.

3.6 Breve revisão e conclusões do capítulo

Neste capítulo foi apresentado o modelo do intervalo de tempo. Este modelo foi criado para suprir uma deficiência na literatura de tempo real em representar uma classe de problemas, entre os quais, os exemplos apresentados no Capítulo 1. Neste ponto, convém explicitar a necessidade de abordagens de escalonamento fazendo uma diferenciação entre a aparente simplidade de executar tarefas sob o modelo do intervalo de tempo e garantir em tempo de projeto que o sistema terá um comportamento esperado.

3.6.1 Infraestrutura de execução x teste de escalonabilidade

Com base no modelo apresentado, pode-se imaginar um cenário no qual existe um sistema de tarefas Γ para ser escalonado num processador. O escalonador usa duas filas: fila de subtarefas prontas (lista ordenada por prioridade) e fila de eventos futuros (lista ordenada por tempo de liberação). Temos inicialmente A_1, A_2, A_3 e A_4 prontas para executar. A subtarefa definida com maior prioridade pelo *EDF* tem direito de usar o processador. Quanto uma subtarefa A_i termina, ela programa a execução de um B_i para um tempo futuro adicionando esta subtarefa na lista de eventos futuros. Quando chegar o momento de liberar B_i, esta ganhará o direito de utilizar o processador, caso o processador não esteja em uso por outra subtarefa B_k que assim ficará na lista de subtarefas aptas. Assume-se que as subtarefas B são não-preemptivas, o que pode ser implementado desabilitando interrupções do processador antes de executar a subtarefa e habilitando as interrupções quando B_i termina. Após terminar sua computação, B_i insere uma subtarefa C_i na lista de subtarefas aptas a executar e sinaliza para o escalonador que sua execução terminou.

Neste exemplo, a infraestrutura de execução é bastante simples. Infelizmente, não é possível saber em tempo de projeto se todas as tarefas são escalonáveis. Mesmo que o sistema de tarefas seja executado durante algum tempo e todas as subtarefas tenham terminado antes de seus deadlines, ainda assim não será um resposta definitiva sobre o comportamento do sistema. Uma situação que ocorre raramente pode não se manifestar em um cenário de execução do sistema de tarefas. A solução para este problema é garantir em tempo de projeto que, no pior caso, o sistema é escalonável.

Uma garantia destas somente pode ser fornecida tendo uma abordagem de escalonamento na qual aspectos como precedência, particionamento de deadlines, liberação de tarefas são analisadas utilizando algumas modelagens como por exemplo *jitters* ou *offsets*. O resultado da abordagem é um método analítico (algoritmo) para realizar um teste sobre o sistema de tarefas. Nos próximos capítulos, serão apresentados os detalhes de cada uma das abordagens em destaque na Figura 3.13, bem como os testes criados e uma avaliação dos resultados.

Capítulo 4

Modo de Execução Não-Preemptivo

Neste capítulo é apresentado o modo de execução não-preemptivo. A característica desta abordagem é o aspecto não-preemptivo das subtarefas B_i que devem executar dentro do intervalo de tempo designado pelas subtarefas A_i. São apresentadas duas técnicas de análise (jitters de liberação e offsets) aplicadas para representar tempos de liberação das subtarefas. Na implementação podem ser utilizados semáforos para representar uma seção de código que não será perturbada ou ainda uma prioridade mais alta do que as demais subtarefas.

4.1 Abordagem com seções não-preemptivas e *jitters*

Neste caso, as subtarefas A_i, B_i e C_i chegam no tempo zero. O sistema de tarefas é síncrono pois todas as subtarefas são liberadas no tempo 0 (mesmo que B e C não possam executar neste ponto). Como visto anteriormente, o tempo em que B_i pode ser liberado é descrito pelo intervalo de tempo $[B_{min}, B_{max}]$. Nesta situação pode-se utilizar *jitters* para modelar, no teste de escalonabilidade o requisito de precedência $A_i \prec B_i \prec C_i$. O *jitter* de uma tarefa representa o máximo tempo em que ela levará desde o tempo em que chega até ser liberada. Em situações reais o *jitter* é utilizado para representar problemas no escalonador como *tick scheduling* (Burns et al., 1995) ou modelar o máximo tempo que uma mensagem leva para ser transmitida. Utilizando-se *jitter* para controlar a liberação das subtarefas B, C e um teste de escalonabilidade que leve o *jitter* em consideração, pode-se verificar através de um teste offline se, para um dado tempo máximo de liberação de B e C, o sistema é escalonável. No caso de uma resposta afirmativa, o sistema de tarefas pode fazer uso de um suporte em tempo de execução, tais como semáforos, mensagens e timers, para liberar as subtarefas em determinado instante.

Nas seguintes seções, a escalonabilidade do sistema de tarefas τ é verificada dividindo-se o problema em duas partes como na Figura 4.1. Na primeira parte, verifica-se a escalonabilidade das subtarefas A_i e C_i na presença de subtarefas não preemptivas B_i cuja liberação é modelada através de *jitters*. Uma resposta negativa (rejeita) significa que o sistema de tarefas τ não é escalonável. Uma resposta positiva (aceita) significa que todas as subtarefas A_i e C_i terminarão até seus deadlines, mesmo sofrendo interferência de subtarefas não-preemptivas.

Na segunda parte, verifica-se a capacidade de B_i executar dentro do seu intervalo de tempo ideal, mesmo que a posição deste possa variar. Esta informação é útil para garantir a execução de tarefas com métrica rígida de *QoS*. Uma resposta negativa significa que o sistema de tarefas não é escalonável. Uma resposta positiva significa que todas as subtarefas B_i rígidas executarão dentro de seus intervalos de tempo ideais, resultando no máximo *QoS*. Além disso, determina-se os valores mínimos e máximos que podem ser obtidos para o *QoS* das subtarefas com métrica de benefício cumulativas.

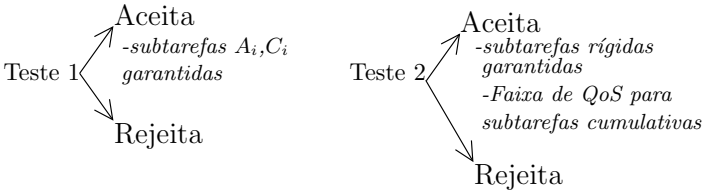


Figura 4.1: Testes de Escalonabilidade.

4.1.1 Teste de escalonabilidade para subtarefas *A* e *C*

Para representar a subtarefa C_i que executa somente após a subtarefa B_i, modela-se a subtarefa C_i com um máximo *jitter* de liberação $J_{C_i} = D_{B_i}$. Assim, B_i possui um *jitter* $J_{Bi} = Bmax_i$. Evidentemente, em algumas situações, B_i pode ser liberada para executar antes deste tempo. A subtarefa A_i possui *jitter* zero.

O teste de escalonabilidade de subtarefas *A* e *C* é realizado utilizando-se a abordagem de demanda de processador (k. Baruah et al., 1990). A demanda de uma tarefa no intervalo de tempo $[t_1,t_2]$ é o tempo cumulativo necessário para processar todas as k instâncias de tarefas que foram liberadas e devem terminar dentro deste intervalo de tempo. Assume-se $g_i(t_1,t_2)$ como a demanda de processamento de τ_i. Desta forma, $g_i(t_1,t_2) = \sum_{r_{i,k}\geq t_1, d_{i,k}\leq t_2} \cdot W_i$. Num sistema de tarefas $\tau = \{\tau_1,\tau_2,\ldots,\tau_n\}$ a demanda de processador em $[t_1,t_2]$ é $g(t_1,t_2) = \sum_{i=1}^{n} g_i(t_1,t_2)$.

A quantidade de tempo de processamento requerido em $[t_1,t_2]$ deve ser menor ou igual ao tamanho do intervalo $[t_1,t_2]$. Assim, $\forall t_1,t_2 : g(t_1,t_2) \leq (t_2 - t_1)$.

Assume-se uma função $\eta_i(t_1,t_2)$ que fornece o número de ativações da tarefa τ_i com liberação e deadline dentro de $[t_1,t_2]$. $\eta_i(t_1,t_2) = max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t1-\Phi_i}{T_i} \rceil\}$. Na Figura 4.2, as únicas ativações contabilizadas por η_i são $\tau_{i,2}$ e $\tau_{i,3}$. A ativação $\tau_{i,1}$ possui um tempo de liberação antes de t_1 e $\tau_{i,4}$ possui um deadline após t_2. O Φ_i representa o offset ou fase de uma tarefa.

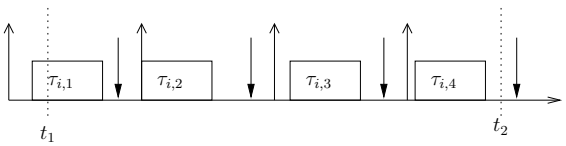


Figura 4.2: Exemplo de ativações de τ_i, computados por η_i

A demanda de processador dentro do intervalo de tempo é igual ao número de ativações que foram liberadas e terminaram dentro do intervalo de tempo multiplicado pelo tempo de computação

W_i. Assim, $g_i(t_1,t_2) = max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t1-\Phi_i}{T_i} \rceil\} \cdot W_i$ e a demanda de processamento para todo o sistema de tarefa é :

$$g(t_1,t_2) = \sum_{i=1}^{n} max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t_1-\Phi_i}{T_i} \rceil\} \cdot W_i \tag{4.1}$$

Simplificado a Equação 4.1 para o caso em que $\Phi = 0$ e utilizando o trabalho (Spuri, 1996), o efeito do *jitter* é inserido na demanda de processador, resultando num novo teste de escalonabilidade:

$$\forall L \geq 0 \qquad \sum_{i=1}^{n} \lfloor \frac{L+T_i+J_i-D_i}{T_i} \rfloor W_i \leq L \tag{4.2}$$

Neste teste, assume-se que as subtarefas possuem deadline menor do que o período e que existe um *jitter* de liberação não nulo para B_i e C_i , enquanto que $J_{A_i} = 0$.

4.1.1.1 Intervalo de teste

Num sistema síncrono, a escala se repete a cada hiperperíodo ($H = mmc(T_1,\ldots,T_n)$). Como o período das subtarefas é o mesmo de sua tarefa original, basta utilizar o período das "n" tarefas para o cálculo. O teste mostrado na Equação 4.2 deve ser realizado até o limite de H. No pior caso do hiperperíodo (onde todos os períodos são números primos entre si) este será o produto de todos os períodos das tarefas.

$$\prod_{i=1}^{n} T_i \tag{4.3}$$

Felizmente, uma inspeção das equações permite reduzir o limite máximo de vezes em que o teste deve ser realizado. Retornando a demanda de processador $g(0,L) = \sum_{i=1}^{n} \lfloor \frac{L+T_i+J_i-D_i}{T_i} \rfloor W_i$, verifica-se que os valores dentro do operador topo $\lfloor \rfloor$ sofrem mudanças sempre que L cruza um deadline d_k e permanece constante até o próximo deadline d_{k+1}. Assim, $g(0,L)$ é uma função degrau (*step*) e somente é necessário avaliar seu valor para os valores de L iguais aos deadlines absolutos em vez de avaliar a cada unidade de tempo.

Uma outra observação permite limitar ainda mais os intervalos a serem verificados (Buttazzo, 2002). Considera-se a existência de uma nova função $G(0,L)$ que representa um limite superior para $g(0,L)$. Fazendo uso de uma propriedade do operador topo, $\lfloor X \rfloor \leq (X)$, ou ainda:

$$\lfloor \frac{L+T_i-D_i+J_i}{T_i} \rfloor \leq (\frac{L+T_i-D_i+J_i}{T_i}) \tag{4.4}$$

Definindo:

$$G(0,L) = \sum_{i=1}^{n} (\frac{L + T_i - D_i + J_i}{T_i}) W_i \tag{4.5}$$

$$G(0,L) = \sum_{i=1}^{n} ((\frac{T_i - D_i}{T_i}) + (\frac{L}{T_i}) + (\frac{J_i}{T_i})) W_i \tag{4.6}$$

$$G(0,L) = \sum_{i=1}^{n} (\frac{T_i - D_i}{T_i}) W_i + \sum_{i=1}^{n} (\frac{L}{T_i}) W_i + \sum_{i=1}^{n} (\frac{J_i}{T_i} W_i) \tag{4.7}$$

$$G(0,L) = \sum_{i=1}^{n} (T_i - D_i) U_i + LU + \sum_{i=1}^{n} (J_i U_i) \tag{4.8}$$

$$G(0,L) = \text{Constante X} + LU + \text{Constante Y} \tag{4.9}$$

$$\text{Constante X + Constante Y = Constante Z} \tag{4.10}$$

$$G(0,L) = \text{Constante Z} + LU. \tag{4.11}$$

$G(0,L)$ é um limite superior para $g(0,L)$ e observa-se que somente um termo da equação $G(0,L)$ é uma função do valor L; os demais serão constantes. Assim, o gráfico desta função será um linha reta como mostrado na Figura 4.3. Quando o tempo $L = 0$, a demanda é igual a $\sum_{i=1}^{n} (T_i - D_i + J_i) U_i$, para $L > 0$ o valor cresce linearmente com coeficiente U (Utilização da tarefa).

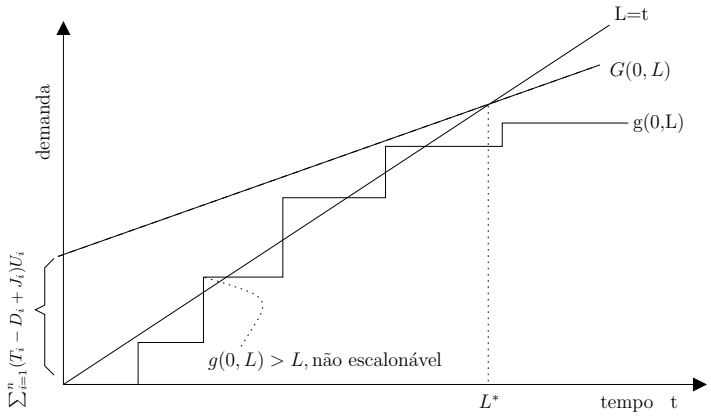


Figura 4.3: Demanda de processador para $g(0,L)$ e $G(0,L)$.

A demanda $G(0,L)$ cruza L (que cresce linearmente com o tempo t) num ponto L^*. Deste ponto em diante L sempre será maior do que $G(0,L)$ e como $g(0,L)$ é sempre menor do que $G(0,L)$ não é necessário verificar a escalonabilidade para valores de $L > L^*$. Calculando o valor de L^* temos:

$$G(0,L^*) = L^* \tag{4.12}$$

$$G(0,L^*) = \sum_{i=1}^{n}(T_i - D_i)U_i + L^*U + \sum_{i=1}^{n}(J_iU_i) = L^* \tag{4.13}$$

$$G(0,L^*) = \sum_{i=1}^{n}(T_i - D_i)U_i + \sum_{i=1}^{n}(J_iU_i) = L^*(1-U) \tag{4.14}$$

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i + \sum_{i=1}^{n}(J_iU_i)}{1-U} \therefore \frac{\sum_{i=1}^{n}(T_i - D_i + J_i)U_i}{1-U}. \tag{4.15}$$

Com esse limite máximo basta selecionar deadlines absolutos $KT_i + D_i$ de forma que o resultado seja menor do que o limite máximo. Formalmente deve-se verificar o teste de escalonabilidade para os deadlines $\beta = \{d_k | d_k = K.T_i + D_i, K.T_i + D_i \leq \frac{\sum_{i=1}^{n}(T_i - D_i + J_i)U_i}{1-U}, \forall K \geq 0, 1 \leq i < n\}$. Além disso, pode-se calcular, no pior caso, quantos serão os deadlines absolutos necessários a verificar. Sendo $d_k = K.T_i + D_i$ e este valor deve ser um inteiro menor ou igual a L^*, temos:

$K \leq \frac{L^* - D_i}{T_i}$

Como procuramos o maior inteiro K e esse número deve representar as ativações K desde $K = 0$, adiciona-se uma unidade. Resultando em

$K_{max} = \lfloor \frac{L^* - D_i}{T_i} \rfloor + 1.$

4.1.1.2 Contabilizando a interferência das subtarefas *B*

Um importante passo para verificar a escalonabilidade de tarefas preemptivas e tarefas não preemptivas foi dado em (Jeffay and Stone, 1993). Os autores mostraram uma condição de escalonabilidade em um modelo para assegurar a escalonabilidade utilizando *EDF* na presença de interrupções. Basicamente, os autores assumem interrupções com tarefas de mais alta prioridade que preemptam qualquer tarefa de aplicação. Desta forma, eles modelam o gerenciador de interrupções como um tempo que é roubado das tarefas de aplicação. Caso as tarefas consigam terminar antes de seus deadlines, mesmo sofrendo a interferência de interrupções, o conjunto de tarefas é escalonável. O conjunto de tarefas é composto por n tarefas de aplicação e m gerenciadores de interrupções. Interrupções são descritas por um tempo de computação CH e um tempo mínimo entre ativações TH. O tempo de processamento para executar interrupções é $f(L)$.

Teorema 4.1.1 *O conjunto τ de n tarefas periódicas ou esporádicas e o conjunto Δ de m gerenciadores de interrupção é escalonável pelo EDF se, e somente se:*

$\forall L \geq 0 \qquad g(0,L) \leq L - f(L), f(L)$ calcula-se

$f(0) = 0$

$$f(L) = \begin{cases} f(L-1)+1, \\ \quad if \sum_{i=1}^{m} \lceil \frac{L}{TH_i} \rceil CH_i > f(L-1) \\ f(L-1), \\ \quad \text{caso contrário} \end{cases} \tag{4.16}$$

A prova deste teorema é similar à prova do método de demanda de processador em (k. Baruah et al., 1990). A diferença está no fato de que, a cada intervalo de tamanho L, a quantidade de tempo que o processador pode dedicar para as tarefas de aplicação é igual a $L - f(L)$.

Utilizando este método, a subtarefa B_i é modelada como um gerenciador de interrupções, A_i e C_i são implementadas como subtarefas executando no *EDF* e a escalonabilidade verificada utilizando-se o Teorema 4.1.1. O Teorema 4.1.1, como descrito por Jeffay e Stone, assume que o sistema é síncrono com um sistema de tarefas no qual os deadlines são iguais aos períodos.

No problema do intervalo de tempo, subtarefas B possuem uma janela de tempo durante a qual podem estar ativas. Aplicar diretamente o Teorema 4.1.1 seria pessimista por contabilizar a influência de interrupções onde elas não poderiam ocorrer.

Algoritmo 1 Escalonabilidade com *jitters* - primeira parte, testa a demanda de processador

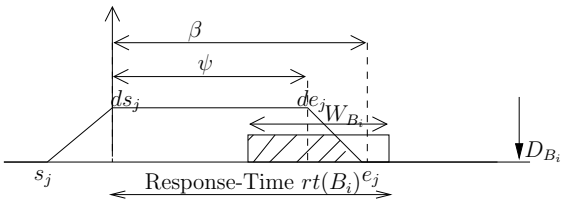
```
for all L such that 0 ≤ L ≤ L* do
  g(0,L) = Σ_{i=1}^{n} (⌊(L+T_i−D_i+J_i)/T_i⌋) · W_i
  if g(0,L) > (L − f(L)) then
    return nonfeasible
  end if
end for
// Escalonável, aplica a segunda parte
return feasible
```

4.1.2 Teste de escalonabilidade para subtarefas B

Diferentemente das subtarefas A_i e C_i que são escalonadas pelo *EDF*, subtarefas B_i são escalonadas com base numa prioridade fixa com valores associados por um algoritmo de associação de prioridades. Considera-se n níveis de prioridade (1,2,...,n), onde n é a prioridade mais baixa. Uma subtarefa B_i é não preemptável por outras subtarefas mas, como possui prioridade maior do que A e C, pode preemptá-las sempre que for liberada.

Pretende-se verificar a escalonabilidade dos B_i, calculando-se seus tempos de respostas rt, assumindo-se que todas as subtarefas B_i são sempre liberadas em ds_j como mostrado na Figura 4.4. Na mesma figura, utiliza-se β para descrever o intervalo de tempo entre a liberação em ds_j até o ponto e_j. Em subtarefas com métrica cumulativa (Figura 3.4) é possível que B_i termine após o intervalo ideal, resultando num baixo valor de *QoS*. Em contraste, subtarefas com métrica rígida (Figura 3.5) demandam

sua execução completa dentro do intervalo ideal. Assim, é necessário verificar se no pior cenário possível $rt(B_i) \leq \psi$. Note que numa subtarefa com métrica rígida B_i, $s_j = ds_j$, $de_j = e_j$.



$rt(B_i)$	QoS
$rt \leq \psi$	100%
$rt \geq \beta + W_{B_i}$	0%
$\psi < rt < \beta + W_{B_i}$	$QoS(B_i, rt(B_i) - W_{B_i}, rt(B_i))\%$

Figura 4.4: *QoS* em relação ao *rt*.

O tempo de reposta pode ser dividido em tempo de resposta de pior caso (*wcrt*) e tempo de resposta de melhor caso (*bcrt*). O *wcrt* representa o pior cenário possível para a execução de B_i e, neste sentido, o *QoS* obtido é o mínimo possível. O *bcrt* representa o melhor cenário possível para B_i resultando no maior *QoS*.

Através do cálculo do *wcrt* e do *bcrt* de uma subtarefa B_i é possível obter um *QoS* como mostrado na Figura 4.4. Desta forma, aplicando o *wcrt* de uma subtarefa B_i como seu tempo de resposta na Figura 4.4 resulta no mínimo *QoS* possível. Aplicando o *bcrt* como seu tempo de resposta, resulta no máximo *QoS* possível. A primeira linha da tabela na Figura 4.4 cobre o caso em que todo B_i executa dentro de seu intervalo de tempo ideal $[ds_j, de_j]$. A segunda linha cobre o caso em que a execução acontece fora do intervalo de tempo $[ds_j, e_j]$ (lembrando que agora considera-se todas as subtarefas B_i liberadas em ds_j) e a terceira linha cobre o caso em que parte de B_i executa dentro do intervalo de tempo $[ds_j, e_j]$. Caso B_i represente uma subtarefa com métrica de qualidade rígida, $rt(B_i)$ deve ser $\leq \psi$ (resultando em *QoS*=100%), caso contrário o *QoS* é $-\infty$ e o sistema de tarefas é rejeitado.

4.1.2.1 Calculando o tempo de resposta - Janela de Tempo

O tempo de resposta de pior caso de tarefas num sistema de prioridade fixa pode ser calculado utilizando o trabalho (Audsley et al., 1993). Fazendo adaptações para o problema específico de subtarefas não-preemptáveis, resulta na seguinte equação composta pela soma de três fatores.

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (W_{B_j}) + \sum_{j \in hp(i)} W_{B_j} \tag{4.17}$$

O primeiro termo na Equação 4.17 é o tempo de execução de pior caso da subtarefa B_i. O segundo termo é o máximo tempo que B_i pode ficar bloqueada por uma subtarefa que se encontra em execução no momento que B_i é liberada. Contabiliza-se este valor como o maior *wcet* entre as subtarefas B_j com prioridade menor (lp) do que B_i, deixando a interferência das subtarefas de prioridade mais alta (hp) para o próximo termo. O último termo é o máximo tempo de bloqueio causado por subtarefas

B_j com prioridades mais altas. Contabiliza-se este valor adicionando todas as subtarefas B_j com prioridade mais alta do que B_i.

Infelizmente, em algumas situações as janelas de tempo nas quais B_i e B_j podem estar ativas podem não se sobrepor. Neste caso, é impossível para B_j produzir interferência sobre B_i mesmo que B_j possua prioridade mais alta. Por exemplo em:

subtask	W	T	$Bmin$	$Bmax$	D	$Prio$
B_i	2	50	10	20	30	1
B_j	5	50	35	45	55	2

As janelas de tempo não se sobrepõem, logo, não existe interferência entre B_j e B_i como mostrado na Figura 4.5 item a). Contudo, se trocarmos os valores de $Bmin$ e $Bmax$ de B_j, tal como em $Bmin_{B_j} = 15, Bmax_{B_j} = 35, D_{B_j} = 45$ as janelas de tempo estarão sobrepostas e existirá interferência entre B_i e B_j para contabilizar, como mostrado na Figura 4.5 item b).

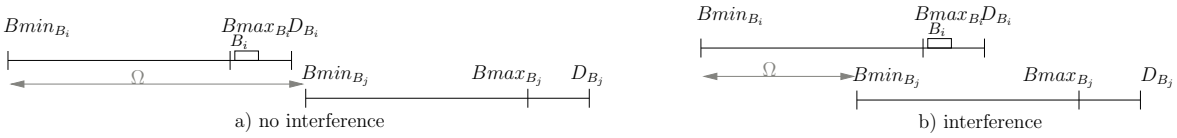


Figura 4.5: Interferência de B_j sobre B_i.

Estende-se a Equação 4.17 para a Equação 4.18 para levar em conta somente as subtarefas que produzem interferência sobre B_i(Algoritmo 2). O Ω na Equação 4.19 fornece o menor intervalo de tempo entre a liberação de B_i e B_j. Se Ω é menor que a distância entre $[D_{B_i}, Bmin_{B_i}]$, as janelas de tempo se sobrepõem, resultando em interferência contabilizada como o tempo de execução de pior caso de B_j. Embora a Equação 4.18 resulte num menor *wcrt* comparando com a Equação 4.17, ela é ainda pessimista no sentido de que a interferência sobre B_i é calculada assumindo-se que os intervalos de tempos entre B_i e B_j são, a cada ativação, os menores possíveis.

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (I_{(B_j,B_i)}) + \sum_{j \in hp(i)} I_{(B_j,B_i)} \tag{4.18}$$

$$\Omega = Bmin_{B_j} - Bmin_{B_i} + \left\lceil \left(\frac{Bmin_{B_i} - Bmin_{B_j}}{gcd(T_{B_i}, T_{B_j})} \right) \right\rceil \cdot gcd(T_{B_i}, T_{B_j}) \tag{4.19}$$

$$bcrt_{B_i} = W_{B_i} \tag{4.20}$$

O tempo de resposta de melhor caso para as subtarefas B_i (mostrado na Equação 4.20) ocorre quando B_i não sofre qualquer interferência de outras subtarefas B_j (por exemplo, quando nenhuma das demais subtarefas B_j é requisitada para executar). Como resultado, o tempo de resposta de melhor caso de B_i é seu próprio tempo de execução de pior caso. Sob a premissa que $W_{B_i} \leq \psi_i$ (Equação 4.4)

Algoritmo 2 Calcula Interferência com *jitters*.

1. **Procedure** $I(B_i, B_j)$
2. `// Compute the interference caused by i upon j.`
3. $interference \leftarrow 0$
4. $d \leftarrow \Omega(B_j, B_i)$
5. **if** $d < D_{B_j}$ **then**
6. **if** $(d < Bmax_j)$**or**$((prio(i) < prio(j))$**and** $(d \geq Bmax_j))$ **then**
7. `// prio(i) < prio(j) In the sense than i has a higher priority than j.`
8. $interference \leftarrow W_{B_i}$
9. **end if**
10. **end if**
11. $d \leftarrow \Omega(B_i, B_j)$
12. **if** $d < D_{B_i}$ **then**
13. $interference \leftarrow W_{B_i}$
14. **end if**
15. **return** $interference$
16. **end procedure**

e que $bcrt_{B_i} = W_{B_i}$, o máximo *QoS* dado pelo teste offline será sempre 100% se o sistema de tarefas é escalonável.

4.1.2.2 Atribuição de prioridade para as subtarefas B

Existem diversas formas de associar prioridades para uma subtarefa de prioridade fixa, as mais conhecidas são o *RM* (Layland and Liu, 1973), (quanto menor o período maior a prioridade) e o *DM* (Leung and Whitehead, 1982), (quanto menor o deadline maior a prioridade). Conectados à atribuição de prioridade existem testes de escalonabilidade. Estes testes verificam se determinado sistema de tarefas, com prioridades associadas sob determinada regra e condições (como $D \leq T$ ou $D = T$), são escalonáveis. Tanto o *RM* ou o *DM* são ótimos no sentido de que, para as condições que eles são aplicáveis ($D = T$ para o *RM* e $D \leq T$ para o *DM*), não existe outro algoritmo de prioridade fixa que seja capaz de escalonar um sistema de tarefas que não possa ser escalonado pelo *RM* ou *DM*.

Uma outra característica de sistemas de prioridade fixa é que, partindo de um instante fixo, a perda do primeiro deadline ocorre na primeira ativação da tarefa. Se a tarefa termina antes do seu primeiro deadline, ela assim o fará para todos os demais deadlines.

Neste trabalho, parte-se para uma outra abordagem em relação à associação de prioridades para as subtarefas B. Ao invés de utilizar-se associação de prioridades pelo *RM* ou *DM*, utiliza-se uma regra heurística com duas métricas **criticalidade** e **fator de deslocamento** e calcula-se o *QoS* obtido pela subtarefa B na pior situação possível.

Na primeira métrica, tarefas podem ser rígidas ou cumulativas. Subtarefas com criticalidade rígida correspondem ao grupo de subtarefas com prioridades mais altas do que tarefas com criticalidade cumulativa. Dentro de cada grupo, prioridades são associadas inversamente ao fator de deslocamento, calculado como $sf_i = \frac{\Psi}{W_{B_i}}$. O fator de deslocamento está relacionado à capacidade da subtarefa de ser

postergada dentro do seu intervalo de tempo ideal e ainda assim obter o máximo *QoS*. Infelizmente, a aplicação da regra heurística não garante um alto valor de *QoS*.

4.2 Abordagem não-preemptiva com *offsets*

Nas seguintes seções, a escalonabilidade do sistema de tarefas τ é verificada dividindo o problema em duas partes como na Figura 4.6. Na primeira parte, verifica-se a escalonabilidade das subtarefas A_i e C_i na presença de subtarefas não preemptivas B_i que podem chegar entre $[Bmin_i, Bmax_i]$. Uma resposta negativa (rejeita) significa que o sistema de tarefas τ não é escalonável. Uma resposta positiva (aceita) significa que todas as subtarefas A_i e C_i terminarão até seus deadlines, mesmo sofrendo interferência de subtarefas não-preemptivas.

Na segunda parte, verifica-se a capacidade de B_i executar dentro do seu intervalo de tempo ideal, mesmo que a posição deste possa variar. Esta informação é útil para garantir a execução de tarefas com métrica rígida de *QoS*. Uma resposta negativa significa que o sistema de tarefas não é escalonável. Uma resposta positiva significa que todas as subtarefas B_i rígidas executarão dentro de seus intervalos de tempo ideais, resultando no máximo *QoS*. Além disso, determina-se os valores mínimos e máximos que podem ser obtidos para o *QoS* das subtarefas cumulativas.

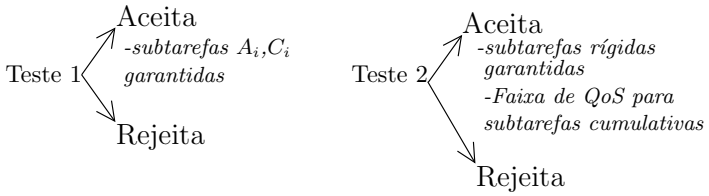


Figura 4.6: Testes de Escalonabilidade.

4.2.1 Teste de escalonabilidade para subtarefas *A* e *C*

O teste de escalonabilidade de subtarefas *A* e *C* é realizado utilizando-se a abordagem de demanda de processador (k. Baruah et al., 1990). A demanda de uma tarefa no intervalo de tempo $[t_1, t_2]$ é o tempo cumulativo necessário para processar todas as k instâncias de tarefas que foram liberadas e devem terminar dentro deste intervalo de tempo. Assume-se $g_i(t_1, t_2)$ como a demanda de processamento de τ_i. Desta forma, $g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} \cdot C_i$. Num sistema de tarefas $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ a demanda de processador em $[t_1, t_2]$ é $g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2)$.

A quantidade de tempo de processamento requerido em $[t_1, t_2]$ deve ser menor ou igual do que o tamanho do intervalo $[t_1, t_2]$. Assim, $\forall t_1, t_2 \; g(t_1, t_2) \leq (t_2 - t_1)$.

Assume-se uma função $\eta_i(t_1, t_2)$ que fornece o número de ativações da tarefa τ_i com liberação e deadline dentro de $[t_1, t_2]$. $\eta_i(t_1, t_2) = max\{0, \lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \rfloor - \lceil \frac{t1 - \Phi_i}{T_i} \rceil\}$. Na Figura 4.7 as únicas

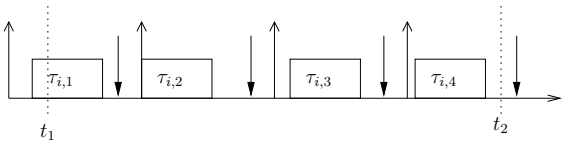


Figura 4.7: Ativações de τ_i

ativações contabilizadas por η_i são $\tau_{i,2}$ e $\tau_{i,3}$. A ativação $\tau_{i,1}$ possui um tempo de liberação antes de t_1 e $\tau_{i,4}$ possui um deadline após t_2.

A demanda de processador dentro do intervalo de tempo é igual ao número de ativações que foram liberadas e terminaram dentro do intervalo de tempo multiplicado pelo tempo de computação C_i. Assim, $g_i(t_1,t_2) = max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t1-\Phi_i}{T_i} \rceil\} \cdot C_i$ e a demanda de processamento para todo o sistemas de tarefa é dada pela Equação 4.21:

$$g(t_1,t_2) = \sum_{i=1}^{n} max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t_1-\Phi_i}{T_i} \rceil\} \cdot W_i \tag{4.21}$$

A escalonabilidade de um sistemas de tarefas assíncrono com deadline menor ou igual ao período pode ser verificado pela Equação 4.22. Em sistemas de tarefas assíncronas, a escala de execução repete-se a cada $[2 \cdot H + \Phi]$ onde H é o hiperperíodo ($H = mmc\{T_1, T_2, \ldots, T_n\}$) e Φ é o maior *offset* entre tarefas ($\Phi = max\{\Phi_1, \Phi_2, \ldots, \Phi_n\}$). Assim, o teste de escalonabilidade deve verificar todos os períodos ocupados em $0, 2 \cdot H + \Phi$, sendo assim de complexidade exponencial $O(H^2)$.

$$\forall t_1, t_2 \qquad g(t_1,t_2) \leq (t_2 - t_1) \tag{4.22}$$

4.2.1.1 Contabilizando a interferência das subtarefas *B*

Um importante passo para verificar a escalonabilidade de tarefas preemptivas e tarefas não preemptivas foi dado por Jeffay e Stone em (Jeffay and Stone, 1993). Os autores mostraram uma condição de escalonabilidade em um modelo para assegurar a escalonabilidade utilizando *EDF* na presença de interrupções. Basicamente, o autor assume interrupções com tarefas de mais alta prioridade que preemptam qualquer tarefa de aplicação. Desta forma, eles modelam o gerenciador de interrupções como um tempo que é roubado das tarefas de aplicação. Caso as tarefas consigam terminar antes de seus deadlines mesmo sofrendo a interferência de interrupções, o conjunto de tarefas é escalonável. O conjunto de tarefas é composto por n tarefas de aplicação e m gerenciadores de interrupções. Interrupções são descritas por um tempo de computação CH e um tempo mínimo entre ativações TH. O tempo de processamento para executar interrupções é $f(L)$.

Teorema 4.2.1 *O conjunto τ de n tarefas periódicas ou esporádicas e o conjunto Δ de m gerenciadores de interrupção é escalonável pelo EDF se e somente se*

$\forall L \geq 0 \qquad g(0,L) \leq L - f(L), f(L)$ calcula-se

$f(0) = 0$

$$f(L) = \begin{cases} f(L-1)+1, \\ \quad if \sum_{i=1}^{m} \lceil \frac{L}{TH_i} \rceil CH_i > f(L-1) \\ f(L-1), \\ \quad \text{caso contrário} \end{cases} \tag{4.23}$$

A prova deste teorema é similar a prova do método de demanda de processador em Baruah. A diferença está no fato de que a cada intervalo de tamanho L, a quantidade de tempo que o processador pode dedicar para as tarefas de aplicação é igual a $L - f(L)$ (Buttazzo and Buttanzo, 1997).

Utilizando este método, a subtarefa B_i é modelada como um gerenciador de interrupções, subtarefas A_i e C_i são implementadas como subtarefas executando no *EDF* e a escalonabilidade verificada utilizando o teorema 4.2.1. O teorema 4.2.1 como descrito por Jeffay e Stone assume que o sistema é síncrono com um sistemas de tarefas no qual os deadlines são iguais aos períodos.

Este teorema é estendido utilizando-se a demanda de processador para representar um sistema assíncrono com deadlines menores ou iguais aos períodos. Neste caso, subtarefas A_i chegam no tempo zero ($\Phi_{A_i} = 0$) e C_i chega no tempo Φ_{C_i}. Neste momento, assume-se que num tempo específico uma interrupção inicia B_i (utilizando a mesma designação de Jeffay e Stone). Para assegurar que a subtarefa C_i somente será executada após a subtarefa B_i, utiliza-se um *offset* $\Phi_{C_i} = D_{B_i}$. O novo teste de escalonabilidade no qual todas as subtarefas C_i possuem *offsets* e subtarefas B_i são modeladas como interrupções é:

$$\forall L \geq 0, \qquad g(t_1,t_2) \leq (t_2 - t_1) - F(t_1,t_2). \tag{4.24}$$

Diferentemente de um sistema síncrono, onde a escalonabilidade pode ser verificada testando-se todos os períodos ocupados L até H(hiperperíodo). Os testes de escalonabilidade para sistemas assíncronos de tarefas possuem alta complexidade algorítmica e torna-se necessário testar todos os períodos ocupados de 0 até $2 \cdot H + \Phi$ (Leung and Merill, 1980).

No problema do intervalo de tempo, subtarefas B possuem uma janela de tempo durante a qual podem estar ativas. Aplicar diretamente o teorema 4.2.1 seria pessimista por contabilizar a influência de interrupções onde elas não poderiam ocorrer. Uma melhoria seria inserir um *offset* ΦH_i em $\lceil \frac{L-\Phi H_i}{TH_i} \rceil$ para representar o fato que uma interrupção não pode ocorrer antes de *Bmin*. No algoritmo 3 assume-se que $F(t_1,t_2)$ representa a demanda de processador resultante das interrupções em $[t_1,t_2]$. No pior caso, o algoritmo possui complexidade $O(H^2)$. Infelizmente, no pior caso, o hiperperíodo é o produto de todos os períodos $\prod_{i=1}^{n} T_i$. Assim, o algoritmo pode ser aplicado apenas quando os períodos do sistema de tarefas resultam num hiperperíodo pequeno.

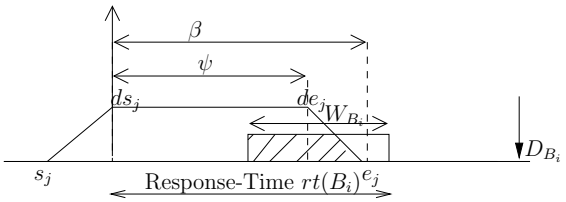
Algoritmo 3 Escalonabilidade com *offsets* - primeira parte

for all t_1 such that $0 \leq t_1 \leq 2 \cdot H + \Phi$ **do**
  for all t_2 such that $t_1 \leq t_2 \leq 2 \cdot H + \Phi$ **do**
    $g(t_1,t_2) = \sum_{i=1}^{n} max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t_1-\Phi_i}{T_i} \rceil\} \cdot W_i$
    $F(t_1,t_2) = f(t_2) - f(t_1)$
    if $g(t_1,t_2) > (t_2 - t_1) - F(t_1,t_2)$ **then**
      return nonfeasible
    end if
  end for
end for
// Escalonável, aplica a segunda parte
return feasible

4.2.2 Teste de escalonabilidade para subtarefas *B*

Diferentemente das subtarefas A_i e C_i que são escalonadas pelo *EDF*, subtarefas B_i são escalonadas com base em prioridade fixa com valores atribuidos por um algoritmo de atribuição de prioridades. Considera-se n níveis de prioridade (1,2,...,n), onde n é a prioridade mais baixa.

Pretende-se verificar a escalonabilidade dos B_i, calculando-se seus tempos de respostas rt, assumindo-se que todas as subtarefas B_i são sempre liberadas em ds_j como mostrado na Figura 4.8. Na mesma figura, utiliza-se β para descrever o intervalo de tempo entre a liberação em ds_j até o ponto e_j. Em subtarefas com métrica cumulativa (Figura 3.4) é possível que B_i termine após o intervalo ideal, resultando num baixo valor de *QoS*. Em contraste, subtarefas com métrica rígida (Figura 3.5) demandam sua execução completa dentro do intervalo ideal. Assim, é necessário verificar se no pior cenário possível $rt(B_i) \leq \psi$. Note que numa subtarefa com métrica rígida B_i, $s_j = ds_j$, $de_j = e_j$.



$rt(B_i)$	QoS
$rt \leq \psi$	100%
$rt \geq \beta + W_{B_i}$	0%
$\psi < rt < \beta + W_{B_i}$	$QoS(B_i, rt(B_i) - W_{B_i}, rt(B_i))\%$

Figura 4.8: *QoS* em relação ao *rt*.

O tempo de reposta pode ser dividido em pior tempo de resposta (*wcrt*) e melhor tempo de resposta (*bcrt*). O *wcrt* representa o pior cenário possível para a execução de B_i e neste sentido o *QoS* obtido é o mínimo possível. O *bcrt* representa o melhor cenário possível para B_i resultando no maior *QoS*.

Através do cálculo do *wcrt* e do *bcrt* de uma subtarefa B_i é possível obter um *QoS* como mostrado na Figura 4.8. Desta forma, aplicando o *wcrt* de uma subtarefa B_i como seu tempo de resposta na Figura 4.8 resulta no mínimo *QoS* possível. Aplicando o *bcrt* como seu tempo de resposta, resulta no

máximo *QoS* possível. A primeira linha da tabela na Figura 4.8 cobre o caso em que todo B_i executa dentro de seu intervalo de tempo ideal $[ds_j, de_j]$. A segunda linha cobre o caso em que a execução acontece fora do intervalo de tempo $[ds_j, e_j]$ (lembrando que agora consideramos todas as subtarefas B_i liberadas em ds_j) e a terceira linha cobre o caso em que parte de B_i executa dentro do intervalo de tempo $[ds_j, e_j]$. Caso B_i represente uma subtarefa com criticalidade rígida, $rt(B_i)$ deve ser $\leq \psi$ (resultando em QoS=100%), caso contrário o *(*QoS) é $-\infty$ e o sistema de tarefas é rejeitado.

4.2.2.1 Calculando o tempo de resposta - Janela de Tempo

O tempo de resposta do pior caso de tarefas num sistema de prioridade fixa pode ser calculado utilizando o trabalho em (Audsley et al., 1993). Fazendo adaptações para o problema específico de subtarefas não-preemptáveis, resulta na seguinte equação composta pela soma de três fatores:

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (W_{B_j}) + \sum_{j \in hp(i)} W_{B_j} \tag{4.25}$$

O primeiro termo na Equação 4.25 é tempo de execução de pior caso da subtarefa B_i. O segundo termo é o máximo tempo que B_i pode ficar bloqueada por uma subtarefa que se encontra em execução no momento que B_i é liberada. Contabiliza-se este valor como o maior *wcet* entre as subtarefas B_j com prioridade menor (lp) do que B_i, deixando a interferência das subtarefas de prioridade mais alta (hp) para o próximo termo. O último termo é o máximo tempo de bloqueio causado por subtarefas B_j com prioridades mais altas. Contabiliza-se este valor adicionando todas as subtarefas B_j com prioridade mais alta do que B_i.

Infelizmente, em algumas situações as janelas de tempo nas quais B_i e B_j podem estar ativos podem não se sobrepor. Neste caso, é impossível para B_j produzir interferência sobre B_i mesmo que B_j possua prioridade mais alta. Por exemplo em:

subtask	*W*	*T*	*Bmin*	*Bmax*	*D*	*Prio*
B_i	2	50	10	20	30	1
B_j	5	50	35	45	55	2

As janelas de tempo não se sobrepõem, logo, não existe interferência entre B_j e B_i como mostrado na Figura 4.9 item *a*. Contudo, se trocarmos $Bmin_{B_j} = 15, Bmax_{B_j} = 35, D_{B_j} = 45$ as janelas de tempo estarão sobrepostas e existirá interferência entre B_i e B_j para contabilizar, como mostrado na Figura 4.9 item *b*.

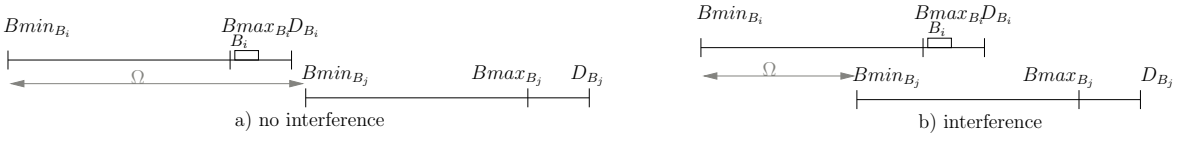


Figura 4.9: Interferência de B_j sobre B_i.

Estende-se a Equação 4.25 para a Equação 4.26 para levar em conta somente as subtarefas que produzem interferência sobre B_i(Algoritmo 4). O Ω na Equação 4.27 dá o menor intervalo de tempo entre a liberação de B_i e B_j. Se Ω é menor que a distância entre $[D_{B_i}, Bmin_{B_i}]$, as janelas de tempo se sobrepõem, resultando em interferência contabilizada como o tempo de execução de pior caso de B_j. Embora a Equação 4.26 resulte num menor *wcrt* comparando com a Equação 4.25, ela é ainda pessimista no sentido de que a interferência sobre B_i é calculada assumindo-se que os intervalos de tempos entre B_i e B_j são a (cada ativação) os menores possíveis.

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (I_{(B_j,B_i)}) + \sum_{j \in hp(i)} I_{(B_j,B_i)} \tag{4.26}$$

Algoritmo 4 Calcula a interferência com *offsets*.

```
1. Procedure I(B_i, B_j)
2. // Compute the interference caused by i upon j.
3. interference ← 0
4. d ← Ω(B_j, B_i)
5. if d < D_{B_j} then
6.    if (d < Bmax_j) or ((prio(i) < prio(j)) and (d ≥ Bmax_j)) then
7.       // prio(i) < prio(j) In the sense than i has a higher priority than j.
8.       interference ← W_{B_i}
9.    end if
10. end if
11. d ← Ω(B_i, B_j)
12. if d < D_{B_i} then
13.    interference ← W_{B_i}
14. end if
15. return interference
16. end procedure
```

$$\Omega = Bmin_{B_j} - Bmin_{B_i} + \left\lceil \left(\frac{Bmin_{B_i} - Bmin_{B_j}}{gcd(T_{B_i}, T_{B_j})} \right) \right\rceil \cdot gcd(T_{B_i}, T_{B_j}) \tag{4.27}$$

$$bcrt_{B_i} = W_{B_i} \tag{4.28}$$

O tempo de resposta de melhor caso para as subtarefas B_i (mostrado na Equação 4.28) ocorre quando B_i não sofre qualquer interferência de outras subtarefas B_j (por exemplo, quando nenhuma das demais subtarefas B_j é requisitada para executar). Como resultado, o tempo de resposta de melhor caso de B_i é seu próprio tempo de execução de pior caso. Sob a premissa que $W_{B_i} \leq \psi_i$ (Figura 4.8) e que $bcrt_{B_i} = W_{B_i}$, o máximo *QoS* dado pelo teste offline será sempre 100% se o sistema de tarefas é escalonável.

4.2.2.2 Atribuição de prioridade para as subtarefas B

Para sistemas de prioridade fixa o *RM* e o *DM* são ótimos no sentido de que não existe outro algoritmo de prioridade fixa que seja capaz de escalonar um sistema de tarefas não escalonável pelo *RM* ou *DM*. O critério de otimalidade assume que todas as tarefas são síncronas, independentes e preemptivas. Quando uma destas premissas é removida, tanto o *RM* quanto o *DM* não são mais ótimos (Audsley, 1991).

Método simples de atribuir prioridades

A forma mais simples de atribuir prioridades é através de uma regra que atribui prioridades baseando-se em critérios estáticos. A regra heurística proposta nesta tese possui duas métricas **criticalidade** e **fator de deslocamento**. Na primeira métrica, tarefas podem ser rígidas ou cumulativas. Subtarefas com criticalidade rígida correspondem ao grupo de subtarefas com prioridades mais altas do que tarefas com criticalidade cumulativa. Dentro de cada grupo, prioridades são atribuidas inversamente ao fator de deslocamento, calculado como $sf_i = \frac{\Psi}{W_{B_i}}$. O fator de deslocamento está relacionado à capacidade da subtarefa de ser postergada dentro do seu intervalo de tempo ideal e ainda assim obter o máximo *QoS*. Como ponto positivo, temos a baixa complexidade computacional deste método $O(n)$. Infelizmente, aplicando a regra heurística não garante um alto valor de *QoS*. Um algoritmo de atribuição de prioridade mais dinâmico atribuiria prioridades em face do *QoS* esperado. Como as interferências são afetadas a cada vez que ajusta-se a prioridade das subtarefas, o algoritmo deveria recalcular as interferências a cada passo.

Atribuição ótima

Em (Audsley, 1991) é mostrado um algoritmo com complexidade $O(n^2)$ para encontrar uma atribuição ótima de prioridade para um sistema de tarefas. Em cada passo, o algoritmo escolhe uma tarefa que deve possuir a prioridade mais baixa disponível e testa se a tarefa é escalonável. Em caso de ser não escalonável, ele escolha outra tarefa e repete o teste. Após encontrar uma tarefa escalonável, o processo é repetido com as tarefas restantes e uma prioridade imediatamente mais alta. Neste caso, uma atribuição ótima é uma atribuição de prioridades para as tarefas tal que todas as tarefas terminem até seus deadlines. Diferentes atribuições de prioridades podem resultar em sistemas de tarefas escalonáveis. Assim, como a única preocupação é a escalonabilidade, diferentes atribuições de prioridade podem resultar em atribuições ótimas.

Diferentemente de (Audsley, 1991) onde o critério de otimalidade é a escalonabilidade e a atribuição de prioridade pode resultar em $\{\text{escalonável}, \text{não escalonável}\}$, no problema do intervalo de tempo o critério de otimalidade está conectado ao H_{prio} (que representa um *QoS* global) e cada atribuição de prioridade pode resultar numa solução diferente. Para o caso específico do trabalho desta tese, a única forma de encontrar uma atribuição ótima de prioridade é enumerar todas os $n!$ possíveis sistemas de tarefas (com prioridades diferentes) onde n é o número de tarefas e escolher uma ou algumas que

resultem na solução ótima. Infelizmente, gerar todos os possíveis sistemas de tarefas com diferentes atribuições de prioridades possui complexidade algorítmica $O(n!)$ o que é proibitivo em muitas aplicações práticas.

No problema do intervalo de tempo, os tempos mínimos para liberar B_i $\{Bmin_1, Bmin_2, \ldots, Bmin_n\}$ caracterizam o escalonamento de subtarefas B_i como um sistema assíncrono B_i. Subtarefas B_i possuem valores de *QoS* e, neste caso, uma atribuições ótima de prioridades deve atribuir prioridades de forma que o sistema de tarefas, como um todo, possua um alto *QoS*. Uma heurística H_{prio} deve ser utilizada como critério para selecionar, entre todas as atribuições de prioridade, a atribuição que é considerada ótima.

Neste trabalho, é apresentada uma heurística para selecionar o sistema de tarefas com a atribuição de prioridade r onde o *QoS* médio $\overline{x}_{QoS_r}$ é alto e o *QoS* de todas as subtarefas apresentem uma pequena dispersão (desvio padrão) s_{QoS_r} em relação à média.

$$\overline{x}_{QoS_r} = \frac{1}{n}\sum_{i=1}^{n} QoS(B_i) \tag{4.29}$$

$$s_{QoS_r} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(QoS(B_i) - \overline{x}_{QoS_r}\right)^2} \tag{4.30}$$

Nesta heurística particular, $H_{prio}(r) \geq H_{prio}(s)$ (no sentido de H_{prio} para uma atribuição de prioridades r é uma solução melhor que H_{prio} para uma atribuição de prioridade s) se e somente se:

$$\overline{x}_{QoS_r} \geq \overline{x}_{QoS_s} \text{ and } \left(\left(s_{QoS_r} \leq s_{QoS_s}\right) \text{ or } \left(s_{QoS_r} \leq \frac{\overline{x}_{QoS_r}}{\overline{x}_{QoS_s}} s_{QoS_s} \right) \right) \tag{4.31}$$

A Equação 4.31 compara os valores de *QoS* médio em ambas atribuições de prioridades r e s. Além disso, ela verifica se a dispersão foi reduzida em r comparado a s ou se, no máximo, aumentou proporcionamente à variação da média dos *QoS*. O Algoritmo 5 apresenta uma solução ótima.

A prova da otimalidade é simplificada pela constatação de que o algoritmo gera todas as permutações possíveis de prioridades. Assume-se que o número de prioridades é o mesmo número n de subtarefas B_i. Para cada atribuição de prioridade para o sistema de tarefas, avalia-se o mesmo pela heurística descrita. As atribuições de prioridade que resultam no maior valor de H_{prio} são consideradas ótimas.

Atribuição subótima

No Algoritmo 6 é apresentado um algoritmo guloso com complexidade $O(n^2)$ para atribuir prioridades para subtarefas B_i. Claramente, o resultado é sub-ótimo no sentido de que não existe garantia que o algoritmo resulte na melhor atribuição de prioridades. O algoritmo encontra a solução escolhendo uma subtarefa q (linha 5) com o *QoS* mais alto (assumindo-se que todas as demais subtarefas

Algoritmo 5 Algoritmo Ótimo de Atribuição de Prioridade.

1. $\{S\} \leftarrow$ all permutations with n priorities
2. $r \leftarrow$ take one priority assignment from $\{S\}$
3. $\{S\} \leftarrow \{S\} - r$
4. **while** $\{S\}$ is not empty **do**
5. $s \leftarrow$ take one priority assignment from $\{S\}$
6. $\{S\} = \{S\} - s$
7. **if** $H_{prio}(r) < H_{prio}(s)$ **then**
8. $r \leftarrow s$
9. **end if**
10. **end while**
11. **return** r

em S possuem prioridades mais altas) para receber a prioridade mais alta disponível. Sempre que uma nova subtarefa B_i (q) é escolhida, as interferências causadas por prioridades mais altas e mais baixas para todas as subtarefas remanescentes que recebem interferência de q são recalculadas. O processo é repetido até que todas as subtarefas possuam prioridades atribuidas. Subtarefas com criticalidade rígida somente podem ser escolhidas (linha 5) quando seus *QoS* forem 100%.

Algoritmo 6 Algoritmo Subótimo de Atribuição de Prioridade.

1. $\{S\} \leftarrow$ all subtasks $B_i \; \forall i \in \{1 \ldots n\}$
2. $p \leftarrow n$ `// Lowest priority.`
3. Compute all interferences$(B_i) \; \forall i \in \{1 \ldots n\}$
4. **while** S is not empty **do**
5. $q \leftarrow$ choose a subtask B_i with the highest *QoS* in $\{S\}$
6. in such way, all B_j have higher priorities
7. $prio(q) = p$
8. `// Assigns the lowest priority available.`
9. $p \leftarrow p - 1$
10. $\{S\} \leftarrow \{S\} - q$
11. **for all** subtasks B_i in $\{S\}$ which interfere with q **do**
12. recompute interferences(B_i)
13. **end for**
14. **end while**

4.2.3 Observações sobre *offsets* e *jitters*

Até este capítulo vimos duas formar para modelar a precedência de tarefas: através de *jitters* e de *offsets*. Independentemente da forma utilizada para implementar a infra-estrutura que será utilizada em tempo de execução, deve-se representar no teste de escalonabilidade a relação de precedência. Assim, utilizam-se testes de escalonabilidade que utilizam *jitters* ou que utilizam *offsets*.

Este capítulo fornece indícios de que a escalonabilidade com *offsets* é superior a um teste de escalonabilidade em que utiliza-se *jitter* para o mesmo fim, mesmo que resulte numa complexidade

algorítmica maior. Abaixo, apresenta-se dois exemplos numéricos simples de um dado sistema de tarefas Γ mostrado na Tabela 4.1 que demonstram essa diferença.

Tarefa	W_i	T_i	D_i	Φ_i/J_i
τ_1	2	10	3	2
τ_2	2	10	3	2
τ_3	2	10	3	2

Tabela 4.1: Exemplo com três tarefas.

Os exemplos das Figuras 4.10 e 4.11 representam do sistema de tarefas Γ composto por três tarefas τ_1, τ_2, τ_3. No primeiro caso (Figura 4.10) temos o sistema utilizando *jitters* para controlar a liberação de duas tarefas. Sobre a mesma figura está representada a demanda de processador (linhas tracejadas) resultante das tarefas. Na primeira figura o sistema não é escalonável, visto que a demanda de processador no tempo 3 é 6 ($W_1 + W_2 + W_3$). Na Figura 4.11 temos o mesmo sistema utilizando-se offsets. A demanda do processador no tempo $t = 3$ é 2, em $t = 5$ é 4 e em $t = 7$ é 6. Sendo assim, a demanda permanece sempre abaixo da linha tracejada L e o sistema é escalonável.

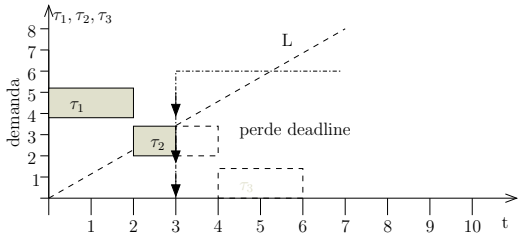


Figura 4.10: Sistema de Tarefas com *Jitter*.

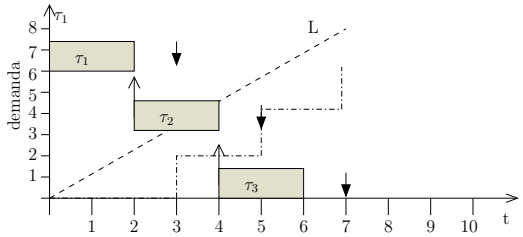


Figura 4.11: Sistema de Tarefas com *Offset*.

O teste de escalonabilidade com *jitters* é pessimista no sentido de que ele irá rejeitar conjuntos de tarefas como não escalonáveis pois considera que todas as subtarefas são liberadas no tempo 0, resultando numa grande demanda de processador. Uma alternativa melhor seria ajustar a chegada das subtarefas τ_2 e τ_3 para um tempo diferente do instante zero através de *offsets*. Neste aspecto, nas próximas abordagens, somente faremos uso de *offsets* para modelar os momentos de chegada das subtarefas.

4.3 Avaliação experimental - modo não preemptivo com *offsets*

Nesta seção apresenta-se o teste de escalonabilidade proposto sobre um sistema não preemptivo de tarefas Γ, comparando seus resultados com uma simulação realizada sobre o mesmo sistema de tarefas. No experimento, Γ é composto por quatro tarefas (τ_1,τ_2,τ_3,τ_4), as quais são divididas em três subtarefas. Os tempos de execução de pior caso, períodos, deadlines, *offsets* e criticalidades são apresentados na Tabela 4.2. Os parâmetros específicos das subtarefas B, tais como ρ,ψ,*Bmin* e *Bmax* são apresentados na Tabela 4.3 e a interferência entre as subtarefas B é representada pelo gráfico da Figura 4.12 com as subtarefas como nós e arestas ligando nós para representar a interferência entre estas subtarefas.

τ	subtarefa	W_i	D_i	T_i	Φ_i	criticalidade
τ_1	A_1	2	6	40	0	
	B_1	4	20	40	7	cumulativa
	C_1	2	40	40	20	
τ_2	A_2	3	9	40	0	
	B_2	3	31	40	9	rígida
	C_2	2	40	40	31	
τ_3	A_3	2	25	80	0	
	B_3	6	38	80	28	cumulativa
	C_3	1	80	80	38	
τ_4	A_4	3	23	120	0	
	B_4	6	35	120	23	cumulativa
	C_4	3	120	120	35	

Tabela 4.2: Exemplo com quatro tarefas.

A simulação da execução do sistema de tarefas foi realizada por 10.000 unidades de tempo, assumindo-se um tempo de liberação uniformemente distribuído entre *Bmin* e *Bmax*. Na simulação, foi arbitrado que as subtarefas B_i e C_i são requisitadas em 90% das ativações τ_i. Neste trabalho, o software de simulação foi desenvolvido especificamente para avaliar o novo modelo de tarefas e coletar os valores de qualidade obtidos durante a execução. As simulações foram realizadas sobre conjuntos de tarefas de tamanhos variados e destes conjuntos, por motivo de clareza, escolheu-se um conjunto com apenas quatro tarefas para ilustrar a avaliação dos métodos desenvolvidos.

No primeiro experimento, todas as subtarefas B_i são assumidas com sua liberação no tempo $t = ds$ e executando sempre com tempo de execução de pior caso. Numa segunda etapa, analisa-se o impacto dessas premissas nos resultados do teste de escalonabilidade em relação ao valor de *QoS* mínimo obtido.

subtarefa	ρ	ψ	Bmin	Bmax
B_1	8	6	6	13
B_2	9	9	9	23
B_3	14	8	25	27
B_4	10	10	23	27

Tabela 4.3: Parâmetros das subtarefas *B*.

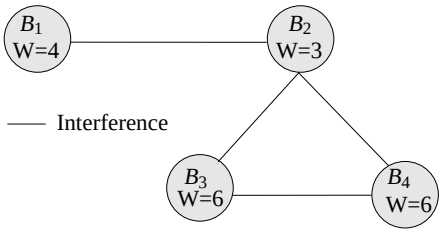


Figura 4.12: Relações de interferência entre subtarefas.

4.3.1 Análises

Nas Tabelas 4.4,4.5,4.6,4.7 são apresentados os passos para atribuir prioridades à subtarefas *B* utilizando o algoritmo 6 onde W_i,max,$\sum$, *QoS* e *prio* representam respectivamente o tempo de execução de pior caso, a máxima interferência por subtarefas de mais baixa prioridade, a soma das interferências de todas as subtarefas de mais alta prioridade, o mínimo *QoS* com uma determinada prioridade e a prioridade da subtarefa em questão.

subtarefa	W_i	*max*	$\sum$	wcrt	*QoS*	prio
► B_1	4	0	3	7	87.5%	4
B_2	3	0	16	19	0.0%	-
B_3	6	0	9	15	11.1%	-
B_4	6	0	9	15	16.6%	-

Tabela 4.4: Escolhe subtarefa B_1.

subtarefa	W_i	*max*	$\sum$	wcrt	*QoS*	prio
B_1	4	0	3	7	87.5%	4
B_2	3	4	12	19	0.0%	-
B_3	6	0	9	15	11.1%	-
► B_4	6	0	9	15	16.6%	3

Tabela 4.5: Escolhe subtarefa B_4.

subtarefa	W_i	*max*	$\sum$	wcrt	*QoS*	prio
B_1	4	0	3	7	87.5%	4
B_2	3	10	6	15	0.0%	-
► B_3	6	6	3	15	11.1%	2
B_4	6	0	9	15	16.6%	3

Tabela 4.6: Escolhe subtarefa B_3.

subtarefa	W_i	*max*	$\sum$	wcrt	*QoS*	prio
B_1	4	0	3	7	87.5%	4
► B_2	3	6	0	9	100.0%	1
B_3	6	6	3	15	11.1%	2
B_4	6	0	9	15	16.6%	3

Tabela 4.7: Escolhe subtarefa B_2.

Na Tabela 4.8 são apresentadas seis possíveis atribuições de prioridades que produzem resultados ótimos (entre estes, a atribuição de prioridade obtida pelo algoritmo 6) dentre o espaço de pesquisa de 24 atribuições para o sistema de tarefas Γ ($n = 4 \therefore 4! = 24$).

Atribuição de Prioridade				$\overline{x}_{QoS_r}$	s_{QoS_r}
B_1	B_2	B_3	B_4		
2	1	3	4	53.81	40.22
2	1	4	3	53.81	40.22
3	1	2	4	53.81	40.222
4	1	2	3	53.81	40.22
3	1	4	2	53.81	40.22
4	1	3	2	53.81	40.22

Tabela 4.8: Seis atribuições de prioridades ótimas para Γ.

O resultado do teste *offline* pode ser visto na Tabela 4.9. A subtarefa B_2 (com criticalidade rígida) sempre executa dentro o intervalo de tempo ideal, resultando no máximo *QoS*. As outras três subtarefas possuem criticalidade cumulativa e possuem um *QoS* mínimo de 87.5%, 11.11% e 16.66% respectivamente. Em virtude de um teste *offline* pessimista, o *wcrt* mostrado na Tabela 4.9 é um limite superior para os valores de *rt* que pode nunca ser obtido na análise por simulação. Assim, deveríamos esperar que os valores reais de *QoS* mínimo (obtido por simulação) fossem mais altos (ou no mínimo iguais) aos valores dados pelo teste *offline*. Da mesma forma, o *bcrt* dado pelo teste *offline* é um limite inferior para o *rt* e está associado com o *QoS* máximo que pode ser obtido. Assim, caso o *rt* simulado seja maior ou igual ao *bcrt* o *QoS* máximo obtido seria menor ou no máximo igual aos valores dados pelo teste *offline*.

Os resultados da simulação mostrados na tabela Tabela 4.10 são consistentes pois os mínimos valores de *QoS* são iguais ou maiores do que os valores dados pelo teste *offline*. Assim, o teste *offline* pode garantir que, durante a execução, não existe a possibilidade de uma subtarefa obter um *QoS* mais baixo do que o valor calculado pelo teste *offline*.

subtarefa	wcrt	bcrt	min QoS	max QoS
B_1	7	4	87.5%	100.0%
B_2	9	3	100.0%	100.0%
B_3	15	6	11.1%	100.0%
B_4	15	6	16.6%	100.0%

Tabela 4.9: Resultados do teste *offline*.

subtarefa	wcrt	bcrt	min QoS	max QoS
B_1	7	4	87.5%	100.0%
B_2	6	3	100.0%	100.0%
B_3	11	6	75.0%	100.0%
B_4	12	6	66.6%	100.0%

Tabela 4.10: Resultados por simulação.

4.3.2 Liberação das subtarefas B_i

Observando os resultados do teste *offline* (Tabela 4.9), percebe-se que algumas subtarefas obtiveram baixos valores de QoS. Uma explicação possível para estes baixos valores seria o momento em que B_i é liberada. Até agora, B_i tem sido liberada em ds. A liberação em ds é útil quando, ainda que no pior caso, a subtarefa em análise consegue terminar dentro do intervalo ideal. Se a mesma subtarefa fosse liberada em $t = s$ e executasse sem interferência, acabaria por receber um baixo valor de QoS, visto que valores altos de QoS são obtidos no intervalo $[ds, de]$. Assim, pode-se analisar em face do tempo de resposta de pior caso (wcrt), qual o melhor momento para liberar B_i tal que o teste *offline* resulte no maior limite inferior para o QoS mínimo.

4.3.2.1 Análise para B_1

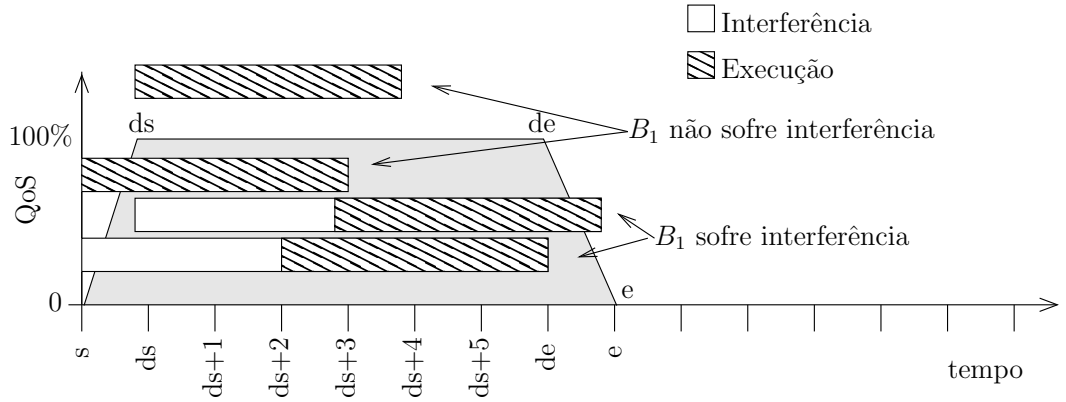


Figura 4.13: Alguns cenários em que B_1 é liberada em $t = s$ e $t = ds$.

Na Figura 4.13 apresenta-se alguns cenários possíveis para a execução de B_1 em função do momento em que esta é liberada. Existem vários momentos em que a subtarefa B_1 pode ser liberada, na figura apresentamos a liberação de B_1 nos tempos $t = s$ e no ds como exemplo. Neste gráfico, deve-se levar em conta que nem sempre B_1 sofrerá a máxima interferência. Para cada um dos tempos de liberação, B_1 pode ser executada imediatamente (numa ativação que não sofre interferência I=0) ou após sofrer interferência. O pior tempo de execução e o tempo de computação são respectivamente 4 e 7 unidades de tempo. Assim, a maior interferência é de 3 unidades de tempo.

Observa-se na figura que quando B_1 é liberada em $t = s$ e não sofre interferência, sua execução (representada pela área com listas inclinadas) inicia antes do intervalo ideal e termina dentro do intervalo ideal resultando num *QoS* baixo pois o máximo *QoS* está dentro do intervalo ideal $[ds, de]$. Quando B_1 é liberada em ds e não sofre interferência, seu *QoS* é máximo, pois toda a sua ativação ocorre dentro do intervalo ideal. No caso em que B_i é liberada em $t = s$ e sofre a máxima interferência, seu *QoS* também será máximo. Já quando é liberada em ds e sofre a máxima interferência, a parte final de sua ativação termina após o intervalo ideal, resultando num baixo *QoS*.

Na Figura 4.14 tem-se um gráfico do *QoS* obtido em função do tempo de liberação de B_1. Para cada instante de tempo (eixo x do gráfico), corresponde um valor de *QoS* (eixo y). A subtarefa é liberada no determinado instante e após o término de sua execução, obtem-se o valor de *QoS*. São apresentadas três curvas. A primeira delas (linha cheia) representa os valores de *QoS* encontrados caso a subtarefa em questão não sofra interferências das demais subtarefas (I=0). No tempo $t = ds + 1.5$, por exemplo, a subtarefa B_1 será liberada 1.5 unidades de tempo após o início de ds. Para esta configuração, e numa situação em que B_1 não sofre interferência (curva de linha cheia), a subtarefa executa integralmente dentro de seu intervalo ideal, obtendo *QoS* de 100%. A curva de linhas e traços representa o *QoS* obtido caso a interferência seja máxima (neste caso, 3 unidades de tempo que resultam no máximo tempo de resposta 7 unidades de tempo). A última curva (tracejada) apresenta uma interferência média (2 unidades de tempo). Pelo gráfico, o tempo ideal para liberar B_1 é $t = s + 0.5$, ponto em que a curva da execução sem interferência corta a curva da execução com a máxima interferência. Neste ponto, encontra-se o maior limite inferior para o *QoS* de B_1. Esse valor é o *QoS* mínimo dado pelo teste *offline*. Pode-se afirmar que mesmo com uma interferência variando de $[0, 3]$ não poderá haver *QoS* menor 96.87% obtido em $t = s + 0.5$ (lembrando que na Tabela 4.9 B_1 sendo liberada em ds resulta em um *QoS* de 87.5%). O gráfico também mostra dois pontos em que o *QoS* obtido é o mesmo liberando B_1 respectivamente em $t = s$ e $t = ds$. Caso a subtarefa sofra uma interferência de 2 unidades de tempo durante sua execução, como mostrado pela curva tracejada, obtem-se um *QoS* maior.

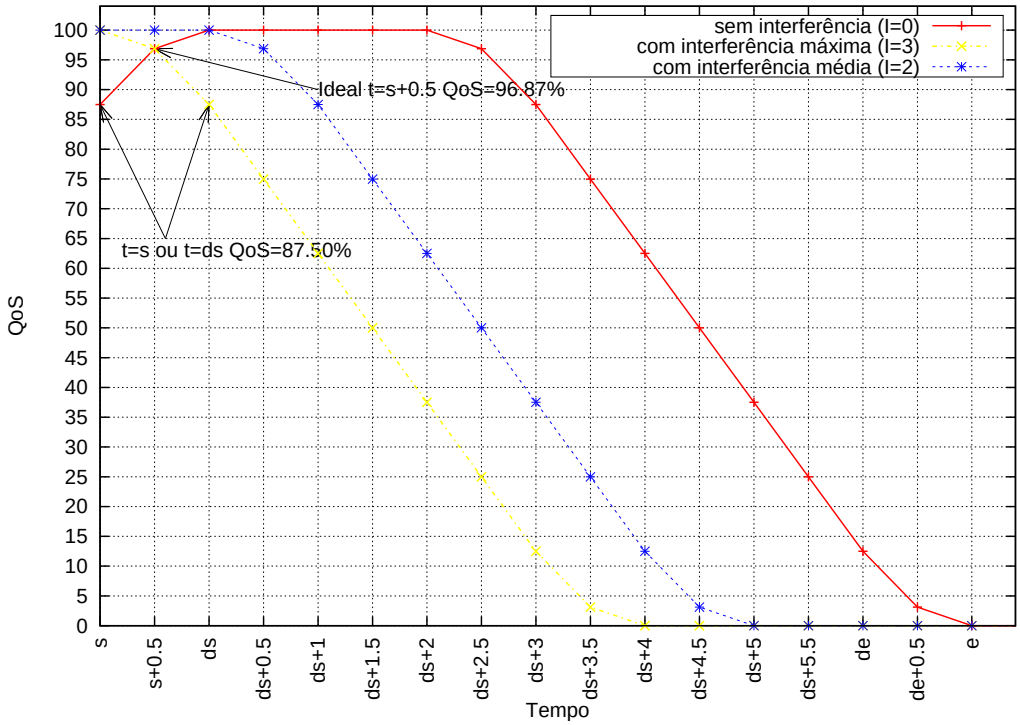


Figura 4.14: QoS em função do tempo de liberação de B_1.

4.3.2.2 Análise para B_2

Como os resultados da Tabela 4.9 mostram a subtarefa B_2 com um *QoS* de 100%, não existe necessidade de uma análise para obter resultados melhores.

4.3.2.3 Análise para B_3

Realizando a mesma análise para B_3, a Figura 4.15 mostra alguns cenários possíveis para a execução de B_3 em função do momento em que esta é liberada. Apresenta-se duas possibilidades: liberar B_3 no tempo $t = s$ ou no tempo ds. Além disso, deve-se que levar em conta que nem sempre B_3 sofrerá a máxima interferência. O pior tempo de execução e o tempo de computação são, respectivamente, 6 e 15. Para cada um dos tempos de liberação, B_3 pode ser executada imediatamente (numa ativação que não sofre interferência I=0) ou após sofrer interferência.

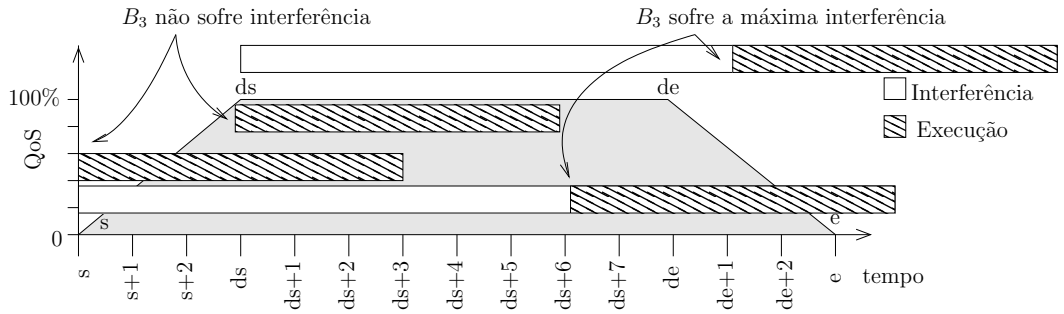


Figura 4.15: Alguns cenários em que B_3 é liberada em $t = s$ e $t = ds$.

Na Figura 4.16 apresenta-se um gráfico do *QoS* obtido em função do tempo de liberação de B_3. São apresentadas três curvas. A primeira delas (linha cheia) representa os valores de *QoS* que encontraríamos caso a subtarefa não recebesse interferências das demais subtarefas. A curva de linhas e pontos representa o *QoS* obtido caso a interferência seja máxima (neste caso, 9 unidades de tempo que resultam no máximo tempo de resposta 15). A última curva (tracejada) apresenta uma interferência pequena (2 unidades de tempo). Pelo gráfico, o tempo ideal para liberar B_3 é $t = s - 0.5$. Neste ponto, tem-se o maior limite inferior para o *QoS* de B_3. Pode-se afirmar que mesmo com uma interferência variando de $[0,9]$ não poderá haver *QoS* menor 66.66% (obtido em $t = s - 0.5$).

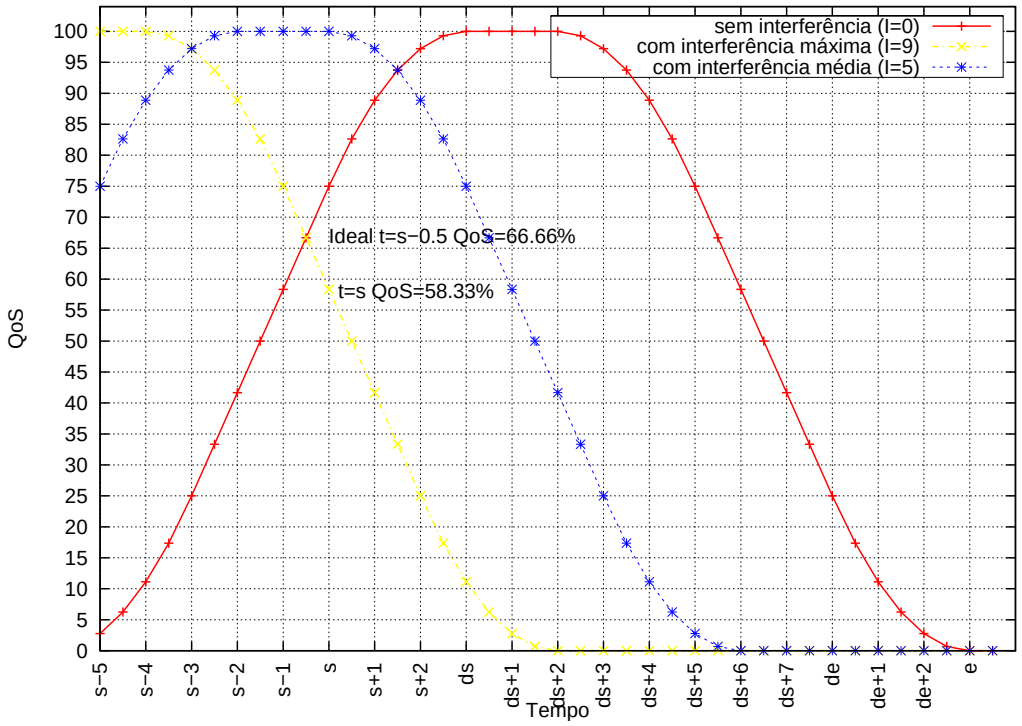


Figura 4.16: QoS em função do tempo de liberação de B_3.

4.3.2.4 Análise para B_4

A subtarefa B_4 possui $s = ds$ e $de = e$. Sendo assim, na Figura 4.17 existe apenas uma situação onde a subtarefa é liberada em $s = ds$ e pode sofrer ou não interferência. Neste caso, a Figura 4.18 serve para demonstrar que quando a tarefa é liberada em $t = s = ds$ seu QoS será, no mínimo, 16.66%. A interferência média representada no gráfico é de 6 unidades de tempo. Uma análise mais detalhada revela que caso B_4 seja liberada antes de $s = ds$, como no tempo $t = s - 2.5$, seu QoS seria aumentado para 58.33%. Ocorre que a principal causa de um QoS baixo é quando a subtarefa recebe a máxima interferência. Assim, liberando B_4 no tempo $t = s - 2.5$, ainda que a subtarefa sofra interferência máxima, conseguirá executar uma parcela maior de seu código dentro do intervalo ideal do que conseguiria caso fosse liberada em $t = s = ds$.

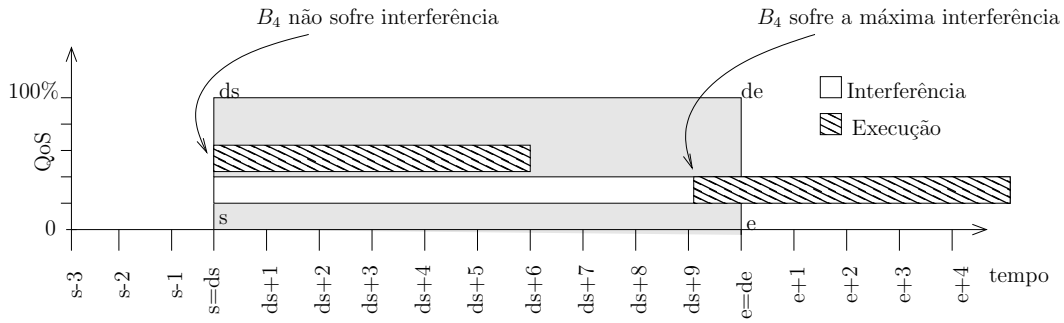


Figura 4.17: Alguns cenários em que B_4 é liberada em $t = s = ds$.

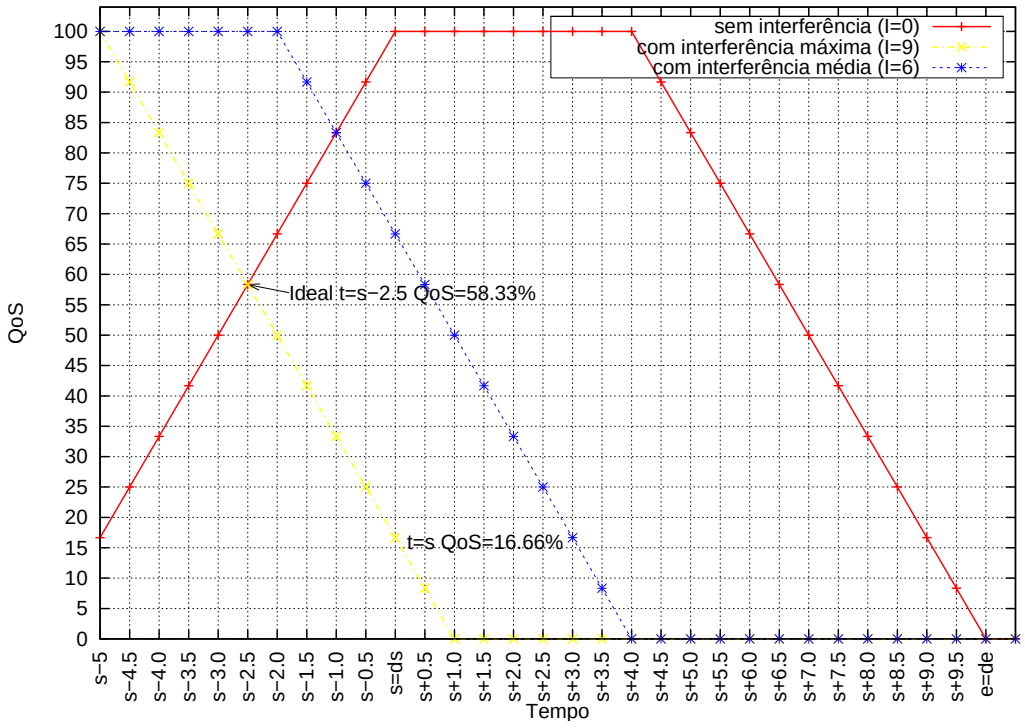


Figura 4.18: QoS em função do tempo de liberação de B_4.

4.3.3 Discussão de resultados

Em face desta análise, temos como novos valores de *QoS* aqueles apresentados na Tabela 4.11. Realizando uma nova simulação (Tabela 4.12) com os novos tempos de liberação para as subtarefas B, obtêm-se valores idênticos aos obtidos pelo teste *offline*.

subtarefa	liberar	wcrt	bcrt	min *QoS*	max *QoS*
B_1	s+0.5	7	4	96.87%	100.0%
B_2	s	9	3	100.0%	100.0%
B_3	s-0.5	15	6	66.66%	100.0%
B_4	s-2.5	15	6	58.33%	100.0%

Tabela 4.11: Novos resultados do teste *offline*.

subtarefa	liberar	wcrt	bcrt	min *QoS*	max *QoS*
B_1	s+0.5	6	4	96.87%	100.0%
B_2	s	8	3	100.0%	100.0%
B_3	s-0.5	11	6	66.66%	100.0%
B_4	s-2.5	12	6	58.33%	100.0%

Tabela 4.12: Novos resultados da simulação.

Infelizmente, a análise para determinar o melhor momento em que B_i deve ser liberado parte do princípio que as subtarefas B_i executam sempre com seu tempo de computação $W_{B_i} = WCET_i$. Quando o $W_{B_i} \leq WCET_i$, a determinação do tempo de liberação pode resultar em valores de *QoS* mínimo menores dos que os informados pelo teste *offline*. O "encurtamento" do tempo de computação faz com que o percentual da tarefa que executa dentro do intervalo de tempo seja menor. A Tabela 4.13 representa esta situação, onde na simulação o tempo de computação das subtarefas B_i é escolhido linearmente entre $W_{B_i}/2+1$ e W_{B_i}. O caso especial é o da subtarefa B_2 para a qual a liberação ocorre no tempo $t = s = ds$ e o encurtamento do tempo de computação não reduz o valor obtido.

subtarefa	liberar	wcrt	bcrt	min *QoS*	max *QoS*
B_1	s+0.5	5	3	95.83%	100.0%
B_2	s	8	2	100.0%	100.0%
B_3	s-0.5	11	4	49.99%	100.0%
B_4	s-2.5	9	4	37.50%	100.0%

Tabela 4.13: Simulação onde os tempos de execução não são constantes.

4.4 Conclusão

Neste capítulo descrevemos o modo de execução não-preemptivo. Para a análise de escalonabilidade, dividimos o problema em duas partes, atacando cada uma delas com modificações de abordagens da literatura de tempo real e através de contribuições novas. Para escalonar as subtarefas A e C utilizamos a demanda de processador (k. Baruah et al., 1990) e modelamos a liberação de C através

de *jitter* ou *offset*. Sobre este tema, ilustra-se, através de um exemplo numérico e argumentação, a vantagem em termos de escalonabilidade dos uso de *offsets*.

Para escalonar as subtarefas B utiliza-se uma adaptação da análise do tempo de resposta para tarefas com prioridade fixa (Audsley et al., 1993). Visto que as atribuições de prioridades dadas por algoritmos clássicos como *RM* e *DM* não são mais ótimas em função das premissas empregadas, apresenta-se três métodos para atribuir prioridades (método símples, ótimo e subótimo) mostrando a complexidade computacional destes. Analisamos o pessimismo para o cálculo da máxima interferência recebida pelas subtarefas B_i através da identificação de quais subtarefas B_j podem causar interferência. Além disso, apresenta-se uma análise que escolhe quais momentos liberar as subtarefas B para melhorar o *QoS*.

Capítulo 5

Modo de Execução Preemptivo

Neste modo de execução, subtarefas B_i podem acessar recursos privados ou recursos de acesso não bloqueante, os quais não precisam tratamento especial para controle de concorrência. O *QoS* obtido pela execução de uma subtarefa B_i será o valor cumulativo da execução do segmento B_i dentro do intervalo de tempo determinado.

5.1 Abordagem preemptiva com *offsets*

Nesta abordagem, subtarefas A e C são escalonadas pelo *EDF* e são utilizados *offsets* para controlar a chegada de B e C, da mesma forma que apresentado no Capítulo 4.

5.1.1 Teste de escalonabilidade para subtarefas A e C

Para verificar a escalonabilidade das subtarefas A_i e C_i num sistema com *offsets* fazendo uso da demanda de processador. Será utilizada a mesma abordagem usada no modo de execução não preemptivo visto no capítulo anterior. A mudança está no fato de que agora a subtarefa B_i é preemptável. Felizmente, como as subtarefas B possuem prioridades mais altas do que as subtarefas escalonadas pelo *EDF*, uma subtarefa B_i somente poderá ser preemptada por uma subtarefa $B_j, prio(i) < prio(j)$. Assim, a análise em relação às subtarefas A e C que sofrem interferência das subtarefas B permanece igual. A mudança existe somente no momento de analisar a escalonabilidade das subtarefas B, através do tempo de resposta.

5.1.2 Teste de escalonabilidade para subtarefas B

No capítulo anterior, calcula-se o tempo de resposta das subtarefas B_i e com base neste valor obteve-se o *QoS* resultante de cada subtarefa. Infelizmente, esta forma de analisar o *QoS* em face do tempo de resposta somente pode ser aplicada para subtarefas não-preemptivas. Na Figura 5.1 (a)

mostra-se uma execução não-preemptiva de um sistema de tarefas no qual o interesse é calcular o *QoS* de B_1.

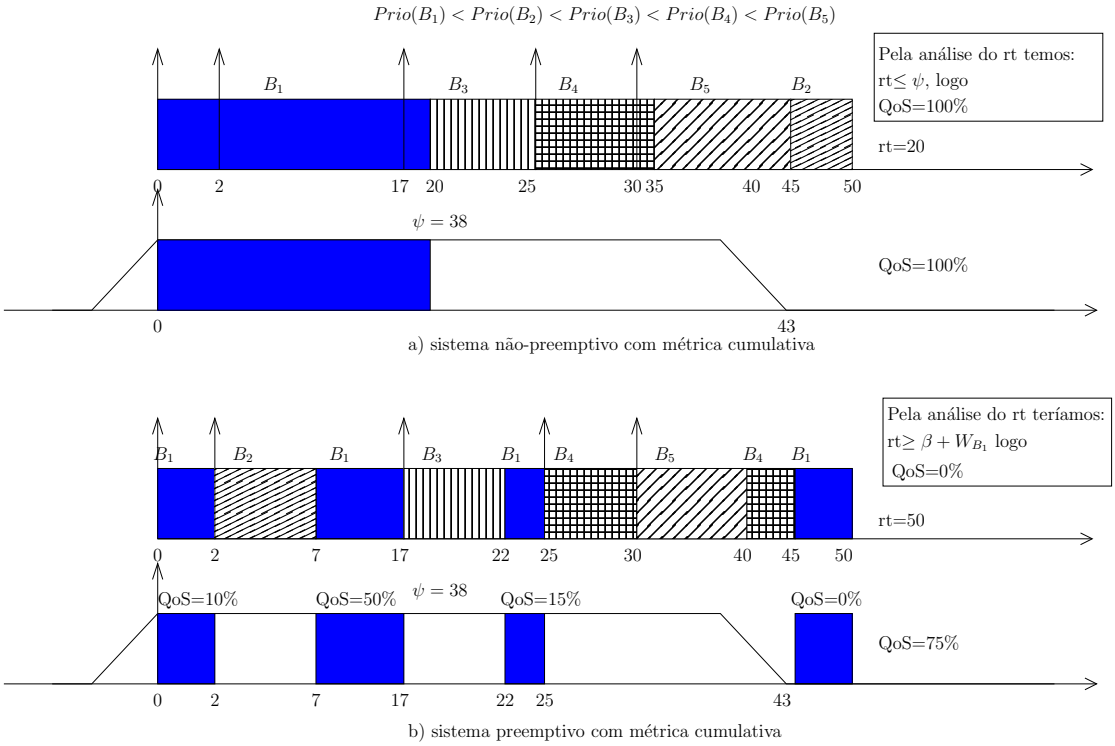


Figura 5.1: Diferenças em Relação ao Tempo de Resposta.

Pelo tempo de resposta de B_1 $rt = 20$ percebe-se que a tarefa obteve *QoS*=100% pois seu $rt \leq \psi$. Já no item b) da mesma figura, $rt = 50$ e caso fosse aplicado o critério do tempo de resposta, *QoS*=0%. Entretanto, B_1 executou dentro do intervalo de tempo ideal nos intervalos $[0,2]$, $[7,17]$ e $[22,25]$, resultando em *QoS* de 10%, 50% e 15%, respectivamente, resultando em *QoS* da subtarefa igual a 75%. Apenas a última parcela de B_1 executou fora de seu intervalo de tempo, sem contribuir para o valor do *QoS*.

5.1.2.1 Calculando o *QoS* para as subtarefas *B*

Na abordagem preemptiva, o teste de escalonabilidade deve contabilizar a execução de cada parcela de B_i, acumulando em cada caso o valor de *QoS* obtido. O somatório destes valores será o QoS_{min} de B_i, desde que as interferências que as demais subtarefas B_j com prioridade maior que B_i sejam escolhidas de forma a resultar na máxima interferência. O resultado do teste *offline* será o resultado da execução de pior caso. Sendo assim, criou-se um algoritmo para calcular o *QoS* de todas as subtarefas *B* do conjunto de tarefas τ.

A idéia básica do algoritmo é determinar quais tarefas B_j interferem com B_i e simular a execução de uma ativação apenas de B_i num cenário de pior caso, onde as subtarefas B_j causam máxima interferência. Como o cenário é pessimista, o valor de qualidade será o mínimo *QoS* possível para B_i. Repete-se esse processo para todas as subtarefas e assim determina-se o mínimo *QoS* para todas as

subtarefas. Dessa forma, o algoritmo é aplicado n vezes, cada uma das quais por uma ativação apenas de B_i.

O primeiro passo do Algoritmo 7 (linhas 1 até 8) é determinar quais são as subtarefas B_j de prioridade mais alta que B_i. Para tal, verifica-se se as tarefas de mais alta prioridade interferem com B_i através da intersecção das janelas de execução. Caso exista intersecção, será possível que num cenário pessimista essas subtarefas B_j estejam prontas para executar ao mesmo tempo de B_i. Desta forma, todas podem ser liberadas para executar ao mesmo tempo. O segundo passo (linhas 9 até 41) é utilizar estas subtarefas B_j num simulador de escalonador com prioridade fixa para contabilizar o *QoS* obtido por B_i. Cada nova ativação k das subtarefas B_j é ajustada para um *arrival-time* (e como não existe *jitter* será o mesmo tempo de liberação) de $(k \cdot T_j + B_{min_j})$. A simulação somente executa por uma ativação de B_i e como o cenário das subtarefas B_j é o que causa mais interferência, o resultado desta simulação será o mínimo valor de *QoS* possível para B_i. Assim, nesta abordagem é calculando o valor de *QoS* cumulativo da subtarefa ao invés do tempo de resposta.

5.2 Avaliação Experimental - modo preemptivo

Nesta seção ilustra-se a efetividade do teste de escalonabilidade proposto sobre um sistema preemptivo de tarefas Γ, comparando seus resultados com uma simulação realizada sobre o mesmo sistema de tarefas. No experimento, Γ é composto por quatro tarefas (τ_1,τ_2,τ_3,τ_4), as quais são divididas em três subtarefas. Os tempos de execução de pior caso, períodos, deadlines, *offsets* e criticalidades são apresentados na Tabela 5.1. Os parâmetros específicos das subtarefas B,(ρ,ψ,*Bmin* e *Bmax*) são apresentados na Tabela 5.2 e a interferência entre as subtarefas B é representada pela figura 5.2 com as subtarefas como nós e arestas ligando nós para representar a interferência entre estas subtarefas.

A simulação da execução do sistema de tarefas foi realizada por 10.000 unidades de tempo, com um tempo de liberação uniformemente escolhido entre *Bmin* e *Bmax*. Na simulação, foi arbitrado que as subtarefas B_i e C_i são requisitadas em 90% das ativações τ_i. Todas as subtarefas B_i têm um tempo de liberação $t = ds$ e são executadas sempre com tempo de execução de pior caso.

subtarefa	ρ	ψ	Bmin	Bmax
B_1	8	6	6	13
B_2	9	9	9	23
B_3	14	8	25	27
B_4	10	10	23	27

Tabela 5.2: Parâmetros das subtarefas B.

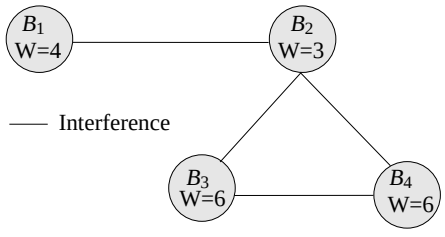


Figura 5.2: Relações de interferência entre subtarefas.

5.2.1 Modo preemptivo com *offsets*

Nas tabelas 5.3 e 5.4 são apresentados os resultados do teste *offline* e da simulação quando a preempção é permitida. São destacados na Tabela 5.4 as diferenças entre os resultados da simulação

Algoritmo 7 Calcula o *QoS* das Subtarefas *B*.

```
1.  for all B_i such that 0 ≤ i ≤ N do
2.    for all B_j such that 0 ≤ j ≤ N and prio(i) < prio(j) do
3.      d_1 = Ω(B_i, B_j)
4.      d_2 = Ω(B_j, B_i)
5.      if d_2 < D_{B_j} or d_1 < D_{B_i} then
6.        // There is interference
7.        insert(B_j,0,ReadyList);
8.      end if
9.      // ReadyList is sorted by priority
10.     // EventList is sorted by release time, assuming jitter zero
11.     left_computation = W_{B_i}
12.     current_time = 0
13.     while (left_computation > 0) do
14.       task ← ReadyList.top()
15.       ReadyList.removeTop()
16.       current_time = max(current_time, task.release_time)
17.       nextEvent_time = EventList.top()
18.       if nextEvent < current_time + task.left_computation) then
19.         // Preempt task
20.         if task = B_i then
21.           QoS_{B_i} ← QoS_{B_i} + QoS(current_time, nextEvent_time) · ((nextEvent_time − current_time)/W_{B_i})
22.           update(task.left_computation)
23.           insert(task, ReadyList)
24.           update(left_computation)
25.         end if
26.       else
27.         if task = B_i then
28.           QoS_{B_i} ← QoS_{B_i} + QoS(current_time, task.left_computation) · ((task.left_computation −
              current_time)/W_{B_i})
29.           update(task.left_computation)
30.           update(left_computation)
31.         else
32.           reRun(task)
33.           // Run again using B_min as the offset
34.         end if
35.       end if
36.       update_global_time(nextEvent_time)
37.       ReadyList ← release_events(EventList)
38.     end while
39.   end for
40. end for
41. // Return the QoS of all subtasks
```

e do teste *offline*, evidenciando que o teste *offline* é pessimista pois resulta em valores menores de *QoS* mínimo em comparação com a simulação. Ainda assim, os valores são bastante próximos, visto que o teste *offline* não é nada mais do que uma simulação com alguns parâmetros ajustados para um cenário pessimista.

5.3 Conclusões

Em comparação com os resultados da seção anterior, observa-se que no modo preemptivo são obtidos valores mais altos de *QoS* do que no modo não-preemptivo. Um fator que tem grande impacto nesta diferença entre valores de *QoS* é a inversão de prioridade que ocorre a todo momento nos sistemas não-preemptivos. Neste capítulo, não utiliza-se a análise que determina o melhor momento

τ	subtarefa	W_i	D_i	T_i	Φ_i	criticalidade
τ_1	A_1	2	6	40	0	
	B_1	4	20	40	7	cumulativa
	C_1	2	40	40	20	
τ_2	A_2	3	9	40	0	
	B_2	3	31	40	9	rígida
	C_2	2	40	40	31	
τ_3	A_3	2	25	80	0	
	B_3	6	38	80	28	cumulativa
	C_3	1	80	80	38	
τ_4	A_4	3	23	120	0	
	B_4	6	35	120	23	cumulativa
	C_4	3	120	120	35	

Tabela 5.1: Exemplo com quatro tarefas.

subtarefa	wcrt	bcrt	min *QoS*	max *QoS*
B_1	7	4	87.5%	100.0%
B_2	3	3	100.0%	100.0%
B_3	9	6	97.2%	100.0%
B_4	15	6	16.6%	100.0%

Tabela 5.3: Resultados do teste *offline*.

subtarefa	wcrt	bcrt	min *QoS*	max *QoS*
B_1	7	4	87.5%	100.0%
B_2	3	3	100.0%	100.0%
B_3	7	6	100.0%	100.0%
B_4	12	6	66.6%	100.0%

Tabela 5.4: Resultados por simulação.

para liberar B_i, visto que a forma com que a interferência se apresenta no modo preemptivo torna sua análise mais complexa. Como nos experimentos realizados todas as subtarefas B são liberadas em $t = ds$, para o caso em que o tempo de computação seja menor que o tempo de computação de pior caso, os resultados serão claramente melhores, diferentemente do que ocorreu no capítulo anterior.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Visão geral do trabalho

Neste trabalho foi apresentado um novo modelo de tarefa para expressar requisitos de tempo real que não podem ser expressos naturalmente em termos de deadlines e períodos. O modelo de tarefas é útil para alguns problemas do mundo real, que pela falta de um estudo teórico, são implementados com escalonadores convencionais, resultando em imprevisibilidade. O modelo faz uso de funções benefício para expressar quando uma ação deveria ser realizada para pbtenção de um benefício máximo. Soluções da literatura, antes inadequadas para o problema, são adaptadas e integradas para o problema do intervalo de tempo.

Foram apresentadas diversas abordagens de escalonamento para o modelo criado, seja síncrono ou assíncrono, com seções não-preemptivas ou preemptivas. A escalonabilidade do sistema é garantida offline através de testes de escalonabilidade criados, os quais, além de uma resposta aceite/rejeita, fornecem um valor de benefício mínimo e máximo para a execução de cada tarefa. São estudados diferentes técnicas para associar prioridades para as tarefas, sendo apresentado um algoritmo para este fim. Além disso, uma análise dos valores de benefício obtidos revela os melhores momentos em que cada tarefa deve ser liberada para obtenção de benefícios máximos.

6.2 Contribuições da Tese

Como contribuições desta tese para a área de tempo real, pode-se mencionar a criação do modelo de tarefas baseado em intervalo de tempo que trata explicitamente um intervalo no qual parte do código de uma tarefa deve ser executada para obtenção de benefício. Anteriomente, utilizando o modelo periódico e os testes de escalonabilidade disponível, não era possível determinar se um sistema de tarefas seria escalonável ou não, muito menos qual o benefício obtido por uma ativação da tarefa em tempo de execução.

Neste trabalho, desenvolveu-se abordagens de escalonamento para o modelo proposto onde foram feitas contribuições na área de algoritmos para associar prioridades para as subtarefas não preemptivas, sem redução do pessimismo na análise do tempo de resposta em face de uma análise mais precisa da interferência causada por outras subtarefas. Finalmente, desenvolveu-se uma análise para determinar o melhor momento para liberar subtarefas B não-preemptiva com o objetivo de obter benefícios maiores. Através das contribuições apresentadas nesta tese, pode-se calcular a escalonabilidade de um sistema de tarefas sob o modelo do intervalo de tempo, obtendo quais o limites mínimos e máximos de QoS para as subtarefas B em tempo de projeto. De posse desta informação, o projetista de sistema pode decidir se alterações no sistema de tarefas são necessárias ou não.

6.3 Perspectivas futuras

Como trabalhos futuros, pretende-se investigar a possibilidade de aumento da qualidade obtida pela execução de subtarefas B_i não-preemptivas através de duas técnicas:

Escalonamento ciente de contexto. Tornar o escalonador ciente do comportamento das tarefas, tal que quando a subtarefa A_i requisita a execução de B_i, ela também informa ao escalonador o quanto de tempo de processamento é necessário para aquela ativação, num cenário em que $W_{B_i} \leq WCET_{B_i}$. Com posse desta informação, o escalonador pode (usando a análise do melhor tempo de liberação de B_i) determinar o melhor tempo de liberação para um W_{B_i}. Num exemplo hipotético, para um B_1 com $W_{B_i} = WCET_{B_i}$, o melhor instante de liberação é $t = s + 0.5$, mas para $W_{B_i} = WCET_{B_i} - 1$, o melhor instante pode ser $t = ds$. Em tempo de execução o escalonador pode decidir liberar B_1 em $t = ds$ em face dessa informação para aumentar o QoS mínimo obtido.

Na mesma linha, a subtarefa A_i pode, ao final de sua execução, informar ao escalonador que não deseja executar B_i. Dessa forma, a interferência que seria causada por uma ativação de B_i sobre outras subtarefas seria eliminada, abrindo possibilidades para melhores escolhas dos tempos de liberação. O escalonamento ciente de contexto ainda que possa melhorar o benefício obtido, este se mostraria presente apenas durante a execução (como um método de refinamento) não sendo possível contabilizar seus ganhos *offline*. Além disso, as operações envolvidas representam um *overhead* durante o tempo de execução.

Reordenação de segmentos não-preemptivos. Durante a execução o escalonador pode violar a ordem de prioridade de duas subtarefas não-preemptivas B_i e B_i fazendo que uma execute antes da outra. A troca é permitida se resultar num incremento de benefício para uma das subtarefas e não interferir no benefício da outra ou, ainda que incorra em perda de benefício, ele seja compensando pelo ganho da anterior segundo alguma métrica.

Referências Bibliográficas

Andrews, D., Welch, L., Chelberg, D., and Brandt, S. (2002). A Framework for Using Benefit Functions in Complex Real-Time Systems. In *Workshop on Parallel and Distributed Real-Time Systems*, pages 92–95.

Audsley, A. N., Burns, A., Richardson, M., and Tindell, K. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292.

Audsley, N. (1991). Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times.

Awerbuch, B., Gawlick, R., Leighton, F. T., and Rabani, Y. (1994). On-line Admission Control and Circuit Routing for High Performance Computing and Communication. In *IEEE Symposium on Foundations of Computer Science*, pages 412–423.

Baker, K. R. and Scudder, G. D. (1990). Sequencing with Earliness and Tardiness Penalties: A Review. *Operations Research*, 38:22–36.

Baker, T. P. (1991). Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99.

Blazewicz, J. (1976). Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines. In *Proceedings of the International Workshop on Modelling and Performance Evaluation of Computer Systems*, pages 57–65. North-Holland.

Burns, A. (1991). Scheduling Hard Real-Time Systems: A Review. *Software Eng. Journal*, 6:116–128.

Burns, A., Prasad, D., Bondavalli, A., Giandomenico, F. D., Ramamritham, K., Stankovic, J., and Strigini, L. (2000). The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. *Journal of Systems Architecture*, 46:305–325.

Burns, A., Tindell, K., and Wellings, A. (1995). Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480.

Burns, A. and Wellings, A. J. (2001). *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Buttazzo, G. C. (2002). *Hard real-time computing systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers.

Buttazzo, G. C. (2005). Rate monotonic vs. edf: judgment day. *Real-Time Syst.*, 29(1):5–26.

Buttazzo, G. C. and Buttanzo, G. (1997). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA.

Buttazzo, G. C., Spuri, M., and Sensini, F. (1995). Value vs. Deadline Scheduling in Overload Conditions. In *IEEE RTSS*, pages 90–99.

Chen, K. and Muhlethaler, P. (1996). A Scheduling Algorithm For Tasks Described By Time Value Function. *Real-Time Systems*, 10(3):293–312.

Chen, M.-I. and Lin, K.-J. (1990). Dynamic priority ceilings: a concurrency control protocol for real-time systems. *Real-Time Syst.*, 2(4):325–346.

Cheng, S., Stankovic, J.-A., and Ramamritham, K. (1988). Scheduling Algorithms For Hard Real-Time Systems: A Brief Survey. *IEEE Computer Society Press*, 1:150–173.

Crespo, A., Ripoll, I., and Albertos, P. (1999). Reducing Delays in RT Control: the Control Action Interval. In 14^{th} *IFAC World Congress on Automatic Control*. Elsevier Science.

Dey, J., J., K., and Towsley, D. (1996). On-line scheduling policies for a class of IRIS (increasing rewardwith increasing service) real-time tasks. In *IEEE Transactions on Computer*, pages 802–813.

Farines, J. M., Fraga, J. d., and Oliveira, R. S. (2000). *Sistemas de Tempo Real*. Escola de Computação 2000.

Garay, J. A., Gopal, I. S., Kutten, S., Mansour, Y., and Yung, M. (1993). Efficient On-Line Call Control Algorithms. In *Israel Symposium on Theory of Computing Systems*, pages 285–293.

Goldman, S. A., Parwatikar, J., and Suri, S. (1997). On-line Scheduling with Hard Deadlines. In *WADS: 5th Workshop on Algorithms and Data Structures*, pages 258–271.

Goossens, J. (2003). Scheduling of Offset Free Systems. *Real-Time Systems*, 24:239–258.

Gutierrez, J. C. P. and Harbour, M. G. (1998). Schedulability Analysis For Tasks With Static And Dynamic Offsets. In *RTSS*, pages 26–37.

Gutierrez, J. P. and Harbour, M. G. (2003). Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF. In *15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 3–12.

Hassin, R. and Shani, M. (2005). Machine scheduling with earliness, tardiness and non-execution penalties. *Computers and Operations Research*, 32:683–705.

Jeffay, K. and Stone, D. L. (1993). Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. In *Proceedings of the* 14^{th} *IEEE Symposium on Real-Time Systems*, pages 212–221.

Jensen, E., Locke, C., and Tokuda, H. (1985). A Time-Driven Scheduling Model for Real-Time Operating Systems.

Jensen, E. D. (1993). A Timeliness Model for Asynchronous Decentralized Computer Systems. In *Autonomous Decentralized Systems*, pages 173–181.

k. Baruah, S., Howell, R. R., and Rosier, L. E. (1990). Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems*, 2:301–324.

Kopetz, H. and Grünsteidl, G. (1994). Ttp-a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23.

Layland, J. and Liu, C. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61.

Leung, J. and Merill, M. (1980). A Note on the Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, 11(3):115–118.

Leung, J. Y. T. and Whitehead, J. (1982). On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2:237–250.

Li, P. (2004). *Utility Accrual Real-Time Scheduling: Models and Algorithms*. PhD thesis, Virginia Polytechnic Institute and State University.

Lipton, R. J. and Tomkins, A. (1994). Online Interval Scheduling. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 302–311, Philadelphia, PA, USA.

Liu, J., Shih, W.-K., Lin, K.-J., Bettati, R., and Chung, J.-Y. (1994). Imprecise computations. In *Proceeding of the IEEE*, volume 82, pages 83–94.

Liu, J. W. S. (2000). *Real-time Systems*. Prentice Hall.

Locke, C. D. (1992). Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives. *Real-Time Systems*, 4(1):37–53.

Mazzini, R. and Armentano, V. A. (2001). A Heuristic For Single Machine Scheduling With Early And Tardy Costs. *European Journal of Operational Research*, 1:129–146.

Mok, A. K. (1983). *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, MIT.

Noergaard, T. (2005). *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes.

Pellizzoni, R. and Lipari, G. (2004). A New Sufficient Feasibility Test For Asynchronous Real-Time Periodic Task Sets. In *Proceedings. 16th Euromicro Conference on Real-Time Systems*, pages 204–211.

Pellizzoni, R. and Lipari, G. (2005). Improved Schedulability Analysis Of Real-Time Transactions With Earliest Deadline Scheduling. In *Proceedings of the 11th IEEE RTAS*, pages 66–75.

Puschner, P. and Koza, C. (1989). Calculating the maximum, execution time of real-time programs. *Real-Time Syst.*, 1(2):159–176.

Ramamritham, K. and Stankovic, J. (1994). Scheduling algorithms and operating systems support for real-time systems. In *Proceedings of the IEEE*, volume 82, pages 55–67.

Ravindran, B., Jensen, E. D., and Li, P. (2005). On Recent Advances In Time/Utility Function Real-Time Scheduling And Resource Management. In 8^{th} *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 55–60.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority Inheritance Protocols: An Approach to Real-Time System Synchronization. In *IEEE Transactions on Computers*, pages 1175–1185. Describes the PCP and PIP for static priorities.

Silberschatz, A. and Galvin, P. (1994). *Operating System Concepts, 4th edition*. Addison-Wesley.

Spuri, M. (1995). *Efficient Deadline Scheduling in Real-Time Systems*. PhD thesis, Scuela Superiore S. Anna.

Spuri, M. (1996). Analysis of Deadline Scheduled Real-Time Systems. Technical report, INRIA, France. RR-2772.

Stankovic, J. (1988). Misconceptions about real-time computing: a serious problem for next-generation systems. *IEEE Computer 21*, 21:10–19.

Stankovic, J. A. (1992). Real-time Computing.

Tindell, K. (1992). Using offset information to analyse static priority pre-emptively scheduled task sets. Technical report, University of York, YCS-92.

Tokuda, H., Wendorf, J. W., and Wan, H. (1987). Implementation of a time-driven scheduler for real-time operating systems. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 271–280.

Velasco, M., Martí, P., and Fuertes, J. M. (2003). The Self Triggered Task Model for Real-Time Control Systems. In *In Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS03)*.

Wu, H., Ravindran, B., Jensen, E. D., and Balli, U. (2004). Utility Accrual Scheduling under Arbitrary Time/Utility Functions and Multi-unit Resource Constraints. In *Proceedings of the* 10^{th} *RTCSA*, pages 80–98.